2018

Offline and Online Density Estimation for Large High-Dimensional Data

Aref Majdara
Michigan Technological University, amajdara@mtu.edu

OFFLINE AND ONLINE DENSITY ESTIMATION FOR LARGE HIGH-DIMENSIONAL DATA

By

Aref Majdara

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2018

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Electrical Engineering.

Department of Electrical and Computer Engineering

Dissertation Advisor:	*Dr. Saeid Nooshabadi*
Committee Member:	*Dr. Daniel Fuhrmann*
Committee Member:	*Dr. Timothy Havens*
Committee Member:	*Dr. Yeonwoo Rho*
Department Chair:	*Dr. Daniel Fuhrmann*

Dedication

To my dear mother Raziyeh and my loving father Iraj,

whose uninterrupted love and support took me where I am today.

Contents

List of Figures

List of Tables

Preface

This doctoral dissertation contains material previously reviewed and published or material submitted to scientific journals and conferences, currently under review. Full citation of these publications are as follows:

Chapter 2:

- Aref Majdara and Saeid Nooshabadi, Nonparametric density estimation using copula transform, Bayesian sequential partitioning and diffusion-based kernel estimator, Paper under review, IEEE Transactions on Pattern Analysis and Machine Intelligence.

- Aref Majdara and Saeid Nooshabadi, Efficient density estimation for high-dimensional data, Paper under review, IEEE Transactions on Knowledge and Data Engineering.

- Aref Majdara and Saeid Nooshabadi, Efficient data structures for density estimation for high-dimensional Big data structures, 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, 2017, pp. 1-4.

Chapter 3:

- Aref Majdara and Saeid Nooshabadi, Block-wise density estimation over high-dimensional data streams, Paper under review, ACM Transactions on Knowledge Discovery from Data.

Chapter 4:

- Aref Majdara and Saeid Nooshabadi, Progressive update of binary partitions for efficient offline and online density estimation, Paper under review, IEEE Transactions on Knowledge and Data Engineering.

Acknowledgments

I would like to thank all those who have helped me learn. Very special thanks to my advisor, Dr. Saeid Nooshabadi for his support and patience throughout my PhD research. I would also like to thank Dr. Daniel Fuhrmann, ECE Department Chair, for the financial support through TA and RA positions. Many thanks to MTU Graduate School for providing financial support for my last semester, through Doctoral Finishing Fellowship.

I would like to thank my wonderful family: my dearest mother Raziyeh, my loving father Iraj, my dear brothers Adel and Abed, my lovely sister Atefeh, my wonderful sister-in-law Neda, and my dear brother-in-law Moahammad, for always giving me their undivided love and support, even from far far away. I would also like to include the youngest member of our family, my sweet and lovely nephew, Behrad, who brought joy and love to our lives. I love you all!

Many thanks to my PhD defense committee members, Dr. Daniel Fuhrmann and Dr. Timothy Havens from Department of Electrical and Computer Engineering, and Dr. Yeonwoo Rho from Department of Mathematical Sciences.

At the end, I would like to thank my dear friend Hossein Tavakoli, for being such a great friend and wonderful roommate during these years.

List of Abbreviations

BBSP	Blockized Bayesian Sequential Partitioning
BP	Binary Partitioning
BSP	Bayesian Sequential Partitioning
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
DPSN	Differential Partition Score Normalized
GPU	Graphical Processing Unit
KDE	Kernel Density Estimation
KL	Kullback-Leibler
KLD	Kullback-Leibler Divergence
KNN	K-Nearest Neighbors
LDA	Linear Discriminant Analysis
MAP	Maximum a-posteriori Probability
MPI	Message Passing Interface
PCA	Principal Component Analysis
PDF	Probability Density Function
PMF	Probability Mass Function
PSN	Partition Score Normalized

SIS	Sequential Importance Sampling
TPW	Tiled Parzen Window
WDE	Wavelet Density Estimator

Abstract

Density estimation has wide applications in machine learning and data analysis techniques including clustering, classification, multimodality analysis, bump hunting and anomaly detection. In high-dimensional space, sparsity of data in local neighborhood makes many of parametric and nonparametric density estimation methods mostly inefficient.

This work presents development of computationally efficient algorithms for high-dimensional density estimation, based on Bayesian sequential partitioning (BSP). Copula transform is used to separate the estimation of marginal and joint densities, with the purpose of reducing the computational complexity and estimation error. Using this separation, a parallel implementation of the density estimation algorithm on a 4-core CPU is presented. Also, some example applications of the high-dimensional density estimation in density-based classification and clustering are presented.

Another challenge in the area of density estimation rises in dealing with online sources of data, where data is arriving over an open-ended and non-stationary stream. This calls for efficient algorithms for online density estimation. An online density estimator needs to be capable of providing up-to-date estimates of the density, bound to the available computing resources and requirements of the application. In response to this, BBSP method for online density estimation is introduced. It works based on collecting and processing the data in blocks of fixed size, followed by a

weighted averaging over block-wise estimates of the density. Proper choice of block size is discussed via simulations for streams of synthetic and real datasets.

Further, with the purpose of efficiency improvement in offline and online density estimation, progressive update of the binary partitions in BBSP is proposed, which as simulation results show, leads into improved accuracy as well as speed-up, for various block sizes.

Chapter 1

Introduction

1.1 Motivation

A variety of modern real-world applications including sensing technologies, security, financial trading, epidemiology, networks and scientific experiments, strongly rely on a proper and timely analysis of stationary or non-stationary streams of data [1] [2]. Density estimation can provide an effective way of obtaining useful insights on important features of the data, such as multimodality and skewness [3]. While density estimation can be used for both continuous and discrete data, it is specially helpful in smoothing continuous data. It can be used as the basis of a range of statistical analyses and machine learning techniques, including non-parametric discriminant

analysis [4], classification, feature analysis [5], cluster analysis [6], bump hunting [7], and anomaly detection [8]. Statistical analysis of big data typically requires analytics in high-dimensional domain, where many of the commonly used techniques fail to perform [9].

Furthermore, with growing availability of data in large volumes, development of computationally efficient data analysis algorithms has become of great importance, in various fields of science and technology. This efficiency can be stated in terms of a number of parameters, including computation time and memory requirements.

In applications with data continuously arriving over a stream, analysis of the data needs to be performed in an online fashion, i.e. the data needs to be processed as it arrives. Efficiency in terms of computation time becomes critical in these online applications. If the rate at which the data is being processed is lower than the arrival rate of data, the processing system will not be able to provide up-to-date results.

This work studies the problem of density estimation in high dimensions, with a focus on developing computationally efficient data structures and algorithms, for offline and online applications.

1.2 Background

Multivariate data analysis is widely used for the purpose of discovering and visualizing hierarchy [10], multivariate image analysis [11], process control [12], biomedicine

[13], bioprocessing [14], etc.

Multivariate density estimation serves as the basis for many of data mining and machine learning techniques. Density estimation is defined as the process of constructing an estimate of the probability density function (PDF), from a set of observed data. For a D-dimensional random vector $\boldsymbol{X} = (X_1, \cdots, X_D)$, its PDF $f_{\boldsymbol{X}}(\mathbf{x})$, where the vector $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_D})$, can be used in computing the probability that point $\boldsymbol{X}$ is located in a certain D-dimensional domain $D = (\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_D)$, is defined as,

$$P(\boldsymbol{X} \in D) = \int_D f_{\boldsymbol{X}}(\mathbf{x}) \mathrm{d}\mathbf{x} \tag{1.1}$$

where $\mathrm{d}\mathbf{x} = \mathrm{d}x_1 \mathrm{d}x_2 \cdots \mathrm{d}x_D$.

Typical parametric density estimation methods [3] [15] [16] assume that the data is coming from a known type of distribution, *e.g.* Normal distribution with mean μ and variance σ^2. Then the process of density estimation aims at estimating the values of theses parameters. However, in high-dimensional problems, parametric methods become largely inefficient, as due to sparsity of data in some areas of the sample space, the number of parameters rapidly increases with the sample size and dimension (*a.k.a. curse of dimensionality*)[15] [17] [18].

Non-parametric methods are more suited for high-dimensional data, as they do not assume any characteristic structure for the data. In these methods, the number of parameters is not fixed, which makes them more flexible than parametric methods.

Kernel density estimation (KDE) [3] [15] [16] [19] [20] [21] is one of the most popular non-parametric density estimators, and is defined by [3],

$$\hat{f}(\mathbf{x}) = \frac{1}{Nh^D} \sum_{i=1}^{N} K\left(\frac{\mathbf{x} - \boldsymbol{X_i}}{h}\right) \tag{1.2}$$

where K is the kernel function, $\boldsymbol{X_i}$ represents the i^{th} sample and h is called the *bandwidth*. In kernel methods, proper choice of the bandwidth is critical, specially in high-dimensional problems.

The most commonly used data-driven method for bandwidth selection is the *plug-in* method [22] [23]. However, this method is found to be negatively affected by the normal reference rule [24] [25] [26].

The work in [26] introduces an adaptive method for kernel density estimation, based on linear diffusion processes and their smoothing properties. They view the kernel of the estimator as the transition density of a diffusion process. In addition, to avoid the negative effects of the normal reference rule, they introduce an improved method for calculating the plug-in bandwidth. Their new nonparametric plug-in method does not require assuming a preliminary Normal model. Main part of their work focuses on one-dimensional problems, but they claim that their method can be extended to higher dimensions.

Some work has been done on density estimation methods using decision trees [27]. In these tree-based methods, the density estimation process involves learning a set of

rules, which are then used for estimating the density for each given test point. In [27], they have presented examples of density estimation for up to 784 dimensions, as well as density-based binary and multi-class classification examples for a 64-dimensional classification problem.

Methods for nonparametric density estimation using wavelets are discussed in [28]. Other works in density estimation include use of weak classifier for density estimation [29] and density estimation based on nearest and farthest neighbor [30].

Multivariate kernel methods [31] [32] are frequently used for multivariate density estimation. The work in [31] presents simulation results for up to 5-dimensional examples. RS-Forest method proposed by [8] provides a fast density estimator for anomaly detection over streams. It uses a forest of randomized space trees (RS-Trees) to partition the data space. In [33], nonparametric multivariate density estimation is performed based on the integration of several multivariate histograms.

In some applications like change point detection in time series [34], estimation of density differences is desired. Some work has been done on direct estimation of density differences [35] [36], and density ratios [37] [38]. Some other works [39] have used an extension of the Pólya Tree [40] to develop a framework for multivariate density estimation. The high computational complexity of this method makes it not a good choice for high-dimensional problems.

Another non-parametric density estimation method is the *histogram* with fixed multi-dimensional bin volume, which is widely used in univariate problems for obtaining a quick estimate of the density function. The basic idea of the histogram can be easily extended to higher dimensions. A multivariate histogram works based on the simple idea of dividing the sample space into equal multi-dimensional *bins* of volume h^D and then counting the number of data points in each bin, to obtain a piecewise constant estimate of the underlying density function. However, with equally spaced bins, it is not possible to adapt to spatially varying smoothness [41]. Even more significantly, in multivariate space, the density function may vary unevenly across the dimensions. To deal with this, various histogram methods with adaptive choice of the bin volume have been proposed [41] [42]. For a variable bin volume histogram with fixed bin size projected along each dimension, the bin volume can be expressed as $h = h_1 h_2 \cdots h_D$. For a more general case of variable bin sizes along each dimension, a D-dimensional bin volume can be expressed as $h_{\mathbf{j}=(j_1,\cdots,j_D)} = \prod_{d=1}^{D} h_{j_d}$, with $1 \leq j_d \leq j_{d_{max}}$. In general $h_{j_d} \neq h_{k_d}$, or more generally $h_{\mathbf{j}} \neq h_{\mathbf{k}}$. For a sample $\boldsymbol{X} = \mathbf{x}$, located in the D-dimensional bin volume of $h_{\mathbf{j}}$, the density is estimated as [3],

$$f(x) = \frac{1}{N} \times \frac{n_{\mathbf{j}}}{h_{\mathbf{j}}} \tag{1.3}$$

where N is the total number of samples and $n_{\mathbf{j}}$ is the data count in bin volume $h_{\mathbf{j}}$. Even with variable bin sizes, the direct application of histogram with regular grid structure becomes impractical as the number of bins required grows exponentially

with the dimension. As an example, for a 10-dimensional test case in MATLAB®, the maximum number of bins that can be allocated before the system runs out of memory is 8^{10}. This corresponds to only eight bin segments along each dimension! To significantly reduce the number of bins, requires partitioning the sample space into irregular partitions of various sizes. The method of adaptive histogram, in which the irregular bin volumes are chosen in a data-dependent way, has been proposed before [41] [42] [43] [44] [45]. Data-dependent *Bayesian sequential partitioning* (BSP) method using sequential importance sampling (SIS) [46] [47] has been proposed [9] as an efficient way of partitioning the sample space, based on Bayesian inference. Binary partitioning of the sample space in a sequential fashion, has also been used in [48], for creating a data-adaptive partition over the sample space. Star discrepancy [49] is used as a measure of uniformity of data distribution, in order to decide which areas of the sample space need further partitioning.

In this dissertation, BSP is used as a basis for developing an efficient algorithm for high-dimensional density estimation and its application is extended to online density estimation for high-dimensional data.

1.3 Outline

The rest of this dissertation is organized as follows. Chapter 2 presents density estimation using BSP algorithm, with a focus on high-dimensional density estimation. It presents the proposed data structures for the efficient implementation of BSP and discusses different implementation-related issues. The simulation results for some example cases are presented, for performance evaluation purposes. This chapter also discusses the computational complexity of the copula-transformed BSP and parallelization of the proposed algorithm using the various features of the algorithm. To show some practical applications of non-parametric density estimation, density-based classification and clustering for low and high-dimensional data are presented.

In Chapter 3, a method for blockized density estimation is proposed, as the basis of an online density estimation framework. It first examines the performance of the Blockized BSP (BBSP) algorithm in offline cases and then extends its application to online cases, including stationary and non-stationary streams. Several examples of synthetic and real data streams are provided to evaluate the capabilities of the proposed method in satisfying the general design criteria for online data mining techniques.

Chapter 4 presents the proposed method for improving the performance of offline

and online density estimation. It is based on the idea of progressive update of the binary partitions. Efficiency of the progressive method is evaluated in both offline and online density estimation. Applying this progressive partitioning method to the basic blockized method introduced in Chapter 3, results in improved estimation accuracy and reduced computation time.

Finally, Chapter 5 contains the concluding remarks and suggestions for future works.

Chapter 2

Density Estimation in High-dimensional Domain

2.1 Binary partitioning

Non-parametric density estimation methods have an appeal in physical sciences, due to the fact that they allow embedding of physical prior belief in the analysis. Further, they provide a straightforward path to obtain predictive distribution, and more generally, spectral inference, by means of posterior draws [50].

Histograms are the simplest form of non-parametric density estimation. In univariate domains, histograms are widely used to obtain an understanding of the overall

shape of distribution of data. In multivariate domain, however, the proper choice of bandwidth becomes more challenging and as the number of bins grows exponentially with the dimensions, multidimensional histograms become highly inefficient. Thus, as an alternative to regular histograms described in Chapter 1, the sample space can be partitioned using a *binary partitioning* (BP) scheme, in which only binary cuts are allowed, i.e. each subregion can only be cut into two smaller subdivisions. The most convenient choice for location of the binary cut would be cutting in the middle of the subregion, i.e., into two equal halves. Figure 2.1 illustrates the mid-point BP scheme, in 2-dimensional space. The method works on the idea of choosing the best partitioning scenario after a certain number of cuts, based on some chosen criterion. However, it is not practically feasible to exhaustively generate all the possible scenarios, because the number of paths grows very rapidly, even in low dimensions. For a D-dimensional sample space, at each level j, there are $(j-1) \times D$ possible ways to cut each of the existing partition subregions. It can be shown that the total number of possible ways to partition the sample space into j BP subregions is $(j-1)! \times D^{(j-1)}$. For the 2-dimensional space shown in Figure 2.1, generation of 50 BP subregions, requires evaluation of 3.4×10^{77} possibilities! Thus, instead of exhaustively creating and examining all possible sample partitions, we need to randomly generate a certain number of sample partitions to maintain a good diversity. In [9], a posterior probability is proposed for this purpose, as part of the BSP method that will be described next.

A rather less common choice for the location of the binary cuts is cutting the subregion at the median point where resulting subregions will have the same number of samples [51], rather than equal volumes. In comparison to the mid-point BP scheme, this scheme requires an additional step of searching for the median of the data, to determine the location of the cut. While the BSP method described in the next subsection is based on mid-point BP scheme, the median-based BP scheme will also be discussed later in Section 2.7.2.

2.2 BSP Algorithm

Consider a D-dimensional dataset with N sample points, expressed as an $N \times D$ matrix. In BSP, the smallest D-dimensional sample space containing the dataset is progressively divided into subregions where the density in each of the divided subregions is estimated by simply counting the number of data points that it contains. The algorithm follows a BP scheme, *i.e.* each cut at a given level j splits one of the subregions into two equal halves.

Figure 2.2 illustrates an example of binary sequential partitioning (BSP) in a 2-dimensional sample space. At $j = 1$, the algorithm starts with the entire sample space. Based on the distribution of data, one of the dimensions is chosen for splitting the sample space using BP. At the beginning of each level $j > 1$, in

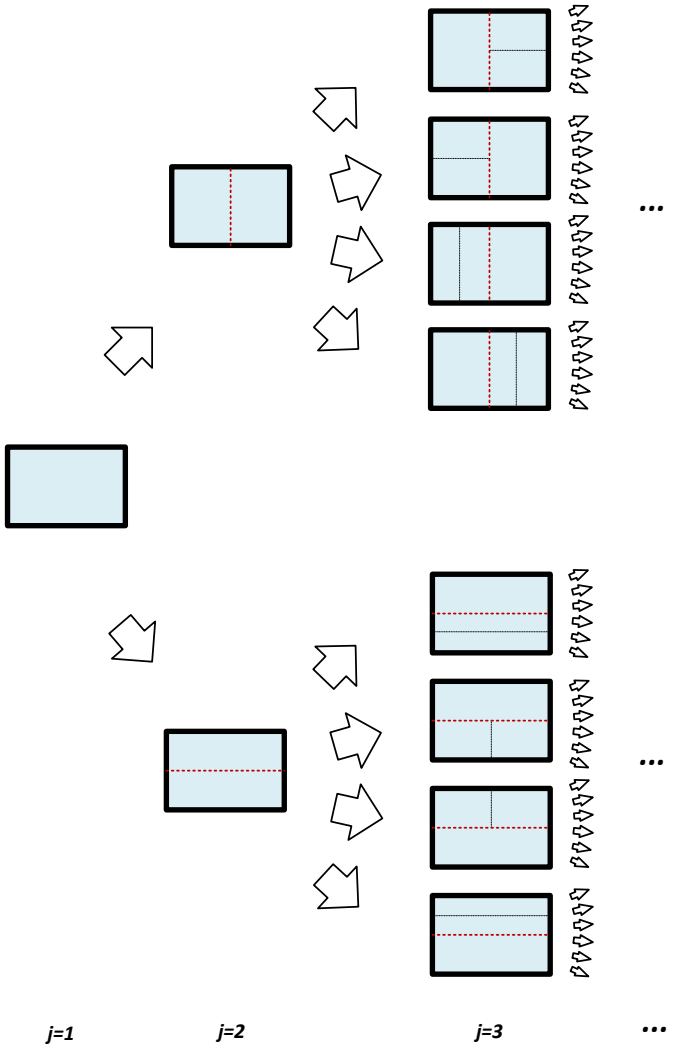


Figure 2.1: Example of mid-point binary partitioning scheme in 2D sample space.

path $g_j = \{cut_2, cut_3, \cdots, cut_{j-1}\}$, the sample space contains $(j-1)$ subregions $(p = 1, \cdots, j-1)$, with subregion p having a volume of v_p, and containing n_p data points. There are $(j-1) \times D$ possibilities for the j^{th} cut, from which one is randomly picked based on some probability mass function as described in [9]. The cuts that lead to the most unbalanced distribution of data in the resulting halves will have the

highest probabilities of being picked. For instance, in the example demonstrated in Figure 2.2, making the first cut in horizontal direction results in point distribution of $n_{(1)} = 12$ and $n_{(2)} = 20$ for two halves. A vertical cut, on the other hand, produces a slightly more unbalanced distribution with $n_{(1)} = 11$ and $n_{(2)} = 21$, and thus, it has the highest probability of being randomly picked. Similar situation prevails in all consequent levels. In Figure 2.2, the dashed lines show all possibilities for the next cut, and the red dashed line shows the one that presents the highest probability.

To improve the quality of density estimation, M independent paths are tried at each level (M sample partitions). At each level j, path ${g_j}^m = \{cut_2^m, cut_3^m, \cdots, cut_{j-1}^m\}$, $(m = 1, \cdots, M)$, the sample space contains $(j-1)$ subregions $(p = 1, \cdots, j-1)$, with subregion p having a volume of v_p, and containing n_p data points. There are $(j-1)\times D$ possibilities for the j^{th} cut. Enumerating the possibilities as $pd = 11, \cdots, 1D, \cdots, (j-1)D$, a conditional probability $s_{j_{pd}}$ is calculated for each of these cuts as [9]:

$$s_{j_{pd}}(cut_{j_{pd}}|g_{j-1}) = C_{j-1}2^{n_p}\frac{\Gamma(n_{pd}^{(1)})\Gamma(n_{pd}^{(2)})}{\Gamma(n_p)} \tag{2.1}$$

where $n_{pd}^{(1)}$ and $n_{pd}^{(2)}$ are data points in each of the resulting halves due to the cut in subregion p along dimension d, and C_{j-1} is a normalizing constant. The sum of all $(j-1)\times D$ conditional probabilities are normalized to unity to construct a probability mass function (PMF), which is used to make the random cut at level j in the chosen subregion p and dimension d, to generate the new subregion $p = j$. Subsequent to the

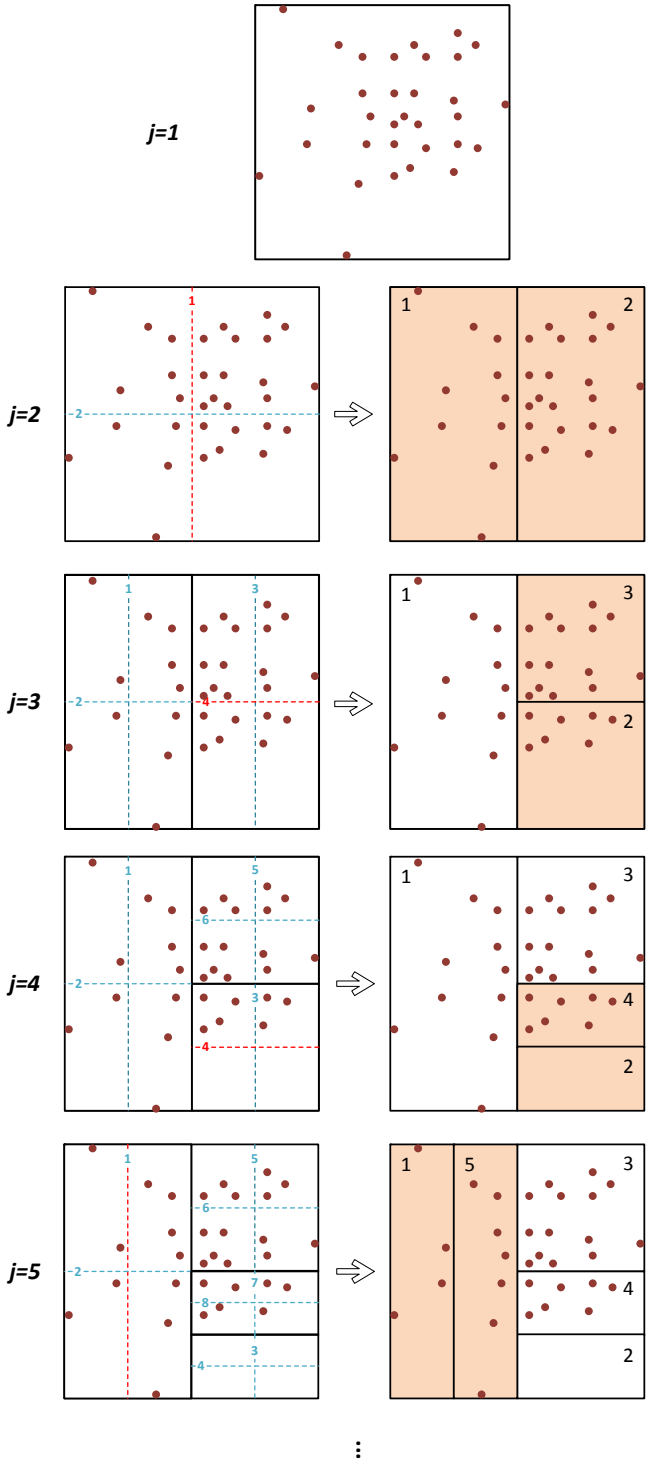


Figure 2.2: Calculating four new cut probabilities at each level, in 2D space.

cut, the data structures holding the information (the number of data points, volumes and coordinates) for the two new subregions, are updated accordingly. This process is repeated until either the best possible partition is obtained, or the number of cuts reaches the maximum value set by the user. Once the optimum partition is obtained, probability density is estimated for each subregion $1 \leq p \leq j$ (a D-dimensional bin), as $n_p/(Nv_p)$. The best partition is the one that gives the lowest error between the actual and estimated densities. However, since the actual density of the data is unknown for real datasets, the algorithm needs to be able to determine the best partition without relying on the knowledge of the actual density.

It has been shown in [9] that the log of the posterior distribution of a sample partition, $\pi(m)$ is a linear function of the Kullback-Leibler divergence (KLD) [36] [52] [53] [54] [55] between the actual and estimated densities, with a negative slope. Thus, in order to minimize the KLD, the algorithm uses the sample partition m with highest $\log(\pi(m))$. To do so, for each sample partition $m \in 1, \cdots, M$, with j levels, a partition *score* is defined as [9]:

$$score(m) = \log(\pi(m)) = -\beta j + \log(\frac{\mathrm{B}(n_1 + \alpha, \cdots, n_j + \alpha)}{\mathrm{B}(\alpha, \cdots, \alpha)}) - \sum_{p=1}^{j} n_p \log(|v_p|) \quad (2.2)$$

where $\alpha \in [0, 0.5]$ and $\beta \in [0.5, 1]$ are two constants. v_p and n_p are the volume and the number of data points in the subregion p, respectively. $\mathrm{B}(u_1, \cdots, u_K)$ denotes the multivariate version of *Beta-function* [56] and is expressed in terms of Γ-*function* as $B(u_1, \cdots, u_K) = \prod_{k=1}^{K} \Gamma(u_k)/\Gamma(\sum_{k=1}^{K} u_k)$. For an update in the value of the

maximum partition score at level j, we stop the BSP algorithm if the score does not improve within a further fixed number of partitioning levels, Δj. A value of $\Delta j = 10$ is used in this work. At this point, partition with maximum score, *i.e. maximum a-posterior* (MAP), is chosen as the best partition.

2.2.1 BSP example

Consider the Gaussian mixture distribution in D dimensions,

$$\boldsymbol{X} \sim \sum_{r=1}^{R} c_r \mathcal{N}_r(\mu_r, \Sigma_r) \tag{2.3}$$

where $\mathcal{N}_r(\mu_r, \Sigma_r)$ is a Normal distribution with the D-dimensional mean vector $\mu_r = (\mu_1, \mu_2, ..., \mu_D)_r$; $\Sigma_r = \text{Cov}[X_i, X_j]_r$ is a $D \times D$ covariance matrix with $i, j = 1, 2, \cdots, D$, and c_r are the mixture weights. Figure 2.3 presents the sample data points for $N = 20,000$ for the case of $R = 3$ and $D = 2$, with the following parameters:

$\boldsymbol{\mu_1} = [2.25, 5.40]$, $\boldsymbol{\mu_2} = [2.60, 5.65]$, $\boldsymbol{\mu_3} = [2.8, 5.15]$,

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.04^2 & 0 \\ 0 & 0.04^2 \end{bmatrix}, \ \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.07^2 & 0 \\ 0 & 0.07^2 \end{bmatrix}, \ \boldsymbol{\Sigma}_3 = \begin{bmatrix} 0.04^2 & 0 \\ 0 & 0.04^2 \end{bmatrix}$$

$c_1 = 0.25, c_2 = 0.4, c_3 = 0.35$.

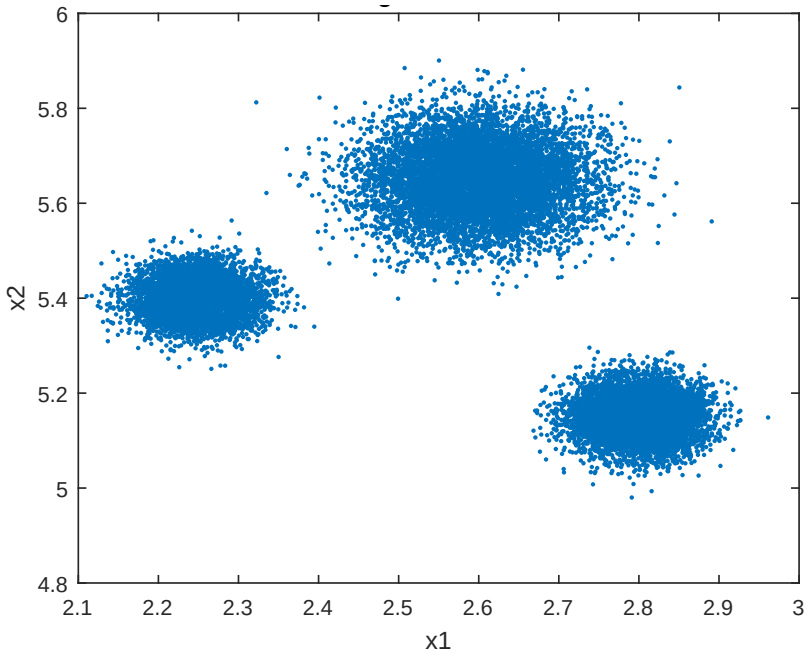


Figure 2.3: Sample data ($N = 20,000$) from a trimodal bivariate normal distribution.

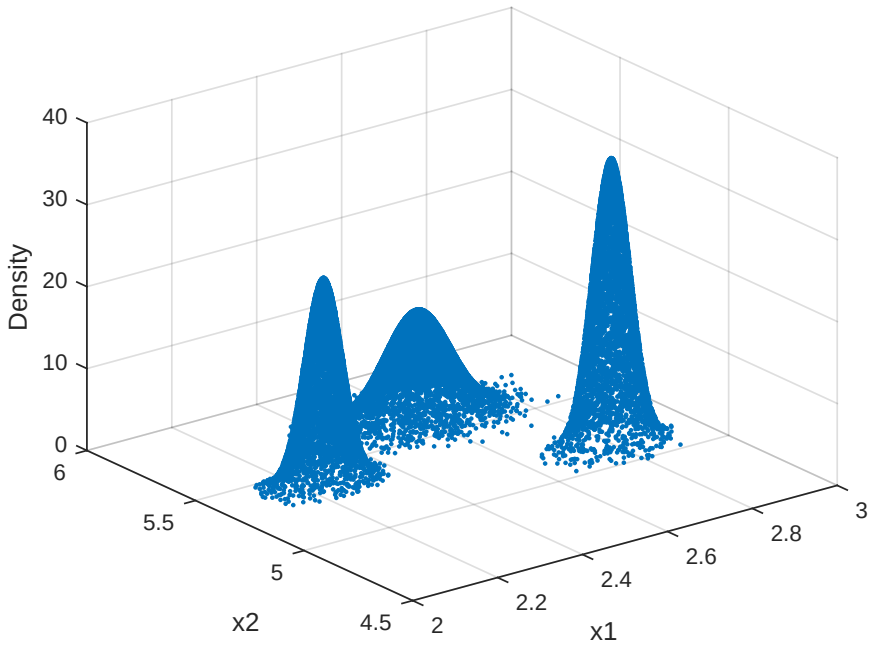


Figure 2.4: Actual joint density of the data in Figure 2.3.

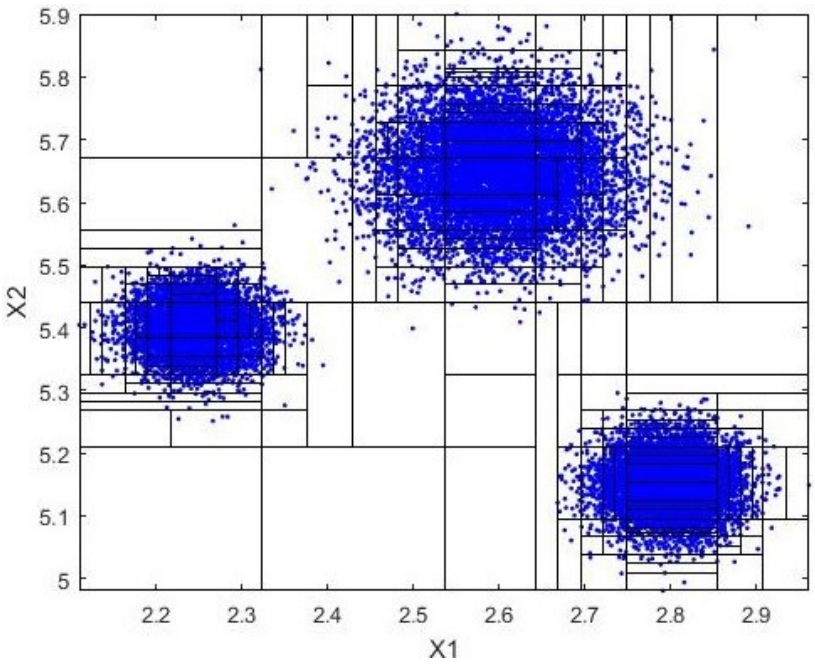


Figure 2.5: BSP cuts on the sample space of Figure 2.3, with $N = 20,000$ and $M = 200$. The number of BSP cuts is 182.

Figure 2.4 presents the actual density of the data in Figure 2.3, obtained from Eq. 2.3. Figure 2.5 presents the result of applying BSP on the sample space of Figure 2.3, with $M = 200$ sample partitions, and $j = 182$ cuts. This simple example clearly demonstrates how the BSP algorithm tends to make more cuts in the areas with more rapidly changing density, and few cuts in the areas with rather uniform distribution. Also, obviously, if a subregion is empty (local density is zero), it will never be further divided into smaller subregions.

2.3 Complexity Reduction Through Copula Transform

Further the true distribution is from uniform, the BSP algorithm requires more cuts to capture the structure of the data. This effect results in dramatic increase in the number of cuts and the deterioration of KLD values for high-dimensional data. The complexity arises due to the fact that in BSP the marginal distributions are learned together with the joint one. To reduce the number of time consuming cuts in the density estimation in high-dimensional space, this section presents utilization of copula transform [41] [57] [58] as a method to map a D-dimensional density estimation problem into the product of D one-dimensional marginal densities and a copula density. Each marginal density is estimated separately, and the results, along with density estimate of the copula, are used to estimate the joint density of the original dataset. The advantage of presenting the density in copula-transformed domain is that the transformed data will have uniform marginal distributions in the interval $[0, 1]$ [41], which leads to a significant reduction in the number of cuts in BSP in high dimensions and much better KLD.

In this work, I am using non-parametric copula, which is estimated directly from the distribution of marginal CDFs. For a random vector $\boldsymbol{X} = (X_1, \cdots, X_D)$, BSP is applied to each of the dimensions, separately, to estimate the marginal PDFs

$f_1(x_1), \cdots, f_D(x_D)$. The estimated PDFs are then used to build the marginal cumulative distribution functions (CDFs), $F_1(x_1), \cdots, F_D(x_D) \in [0,1]$, where $F_d(x_d) = P(X_d \leq x_d)$. The resulting random variables $F_1(X_1), \cdots, F_D(X_D)$ form the new multivariate dataset. As a result, the joint CDF of the sample dataset $F_{\boldsymbol{X}}(\mathbf{x})$ can be expressed as a standard copula C [41] as,

$$F_{\boldsymbol{X}}(\mathbf{x}) = C(F_1(x_1), \cdots, F_D(x_D)) \tag{2.4}$$

In copula domain, instead of using N samples of $\boldsymbol{X} = (X_1, \cdots, X_D)$ to perform the BSP, we use N samples of the generated marginal CDFs as a copula-transformed dataset $(F_1(X_1), \cdots, F_D(X_1))$ (a new $N \times D$ dataset). The method of BSP is then applied to this new D-dimensional dataset, to estimate the joint PDF $c(F_1(x_1), \cdots, F_D(x_D))$, where $c(\mathbf{u}) = \frac{\partial^D}{\partial u_1 \ldots \partial u_D} C(\mathbf{u})$. The PDF for the original dataset, $f(\mathbf{x})$, can then be calculated as [41],

$$f(\mathbf{x}) = c(F_1(x_1), \cdots, F_D(x_D)) \prod_{d=1}^{D} f_d(x_d) \tag{2.5}$$

Simulations show that in high dimensions, use of copula transform reduces the total number of cuts. More importantly, it reduces the number of computationally complex cuts required by the BSP algorithm in high dimensional space by as much as 98%, and substitutes them by computationally cheaper cuts in the marginal distributions.

Table 2.1

BSP with direct and copula-transformed cuts on D-dimensional space for various dimensions, for $N = 20,000$, $M = 200$.

Time unit is seconds. The number of cuts are shown as the sum of total marginal and copula cuts, as well as a pair in the parenthesis corresponding to each component.

	$D = 2$		$D = 32$		$D = 64$	
	Direct	**Copula**	**Direct**	**Copula**	**Direct**	**Copula**
Cuts	182	183 (96+87)	2938	1694 (1614+80)	3648	3338 (3266+72)
Time	13	19	2200	278	6018	552
KLD	0.045	0.018	33.5	0.19	72.5	0.31

Table 2.1 presents the number of cuts, the execution times and the KLD values of the direct and copula-transformed BSP for various dimensions, for a multivariate normal distribution. The synthetic dataset used in these simulations is a trimodal normal distribution for the first two dimensions, as shown in Figure 2.4. In 32 and 64-dimensional datasets, the third dimension is a unimodal normal distribution, and dimensions four and higher have a bimodal normal distribution. This choice of datasets ensures adequate level of diversity in the marginal distributions. Significant from the data in Table 2.1 is the vast disparity between the KLD values for the direct and copula techniques for the high dimensional datasets.

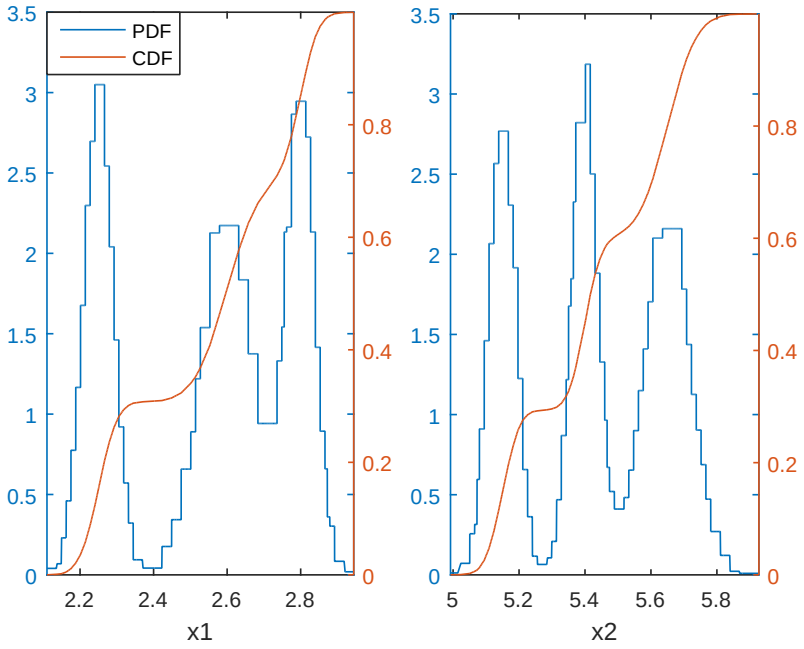


Figure 2.6: Estimated marginal PDF and CDF, using copula transform, for $N = 20,000$ and $M = 200$.

The number of BSP cuts for the two marginals X_1 and X_2 are 49 and 47, respectively.

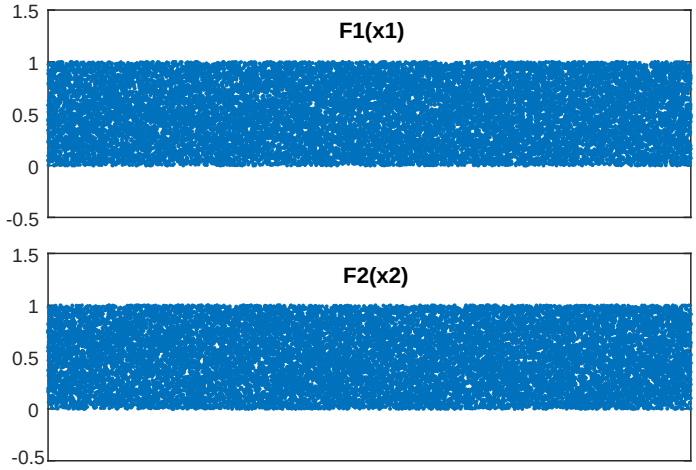


Figure 2.7: Distribution of the transformed random variables $F_1(X_1)$ and $F_2(X_2)$, with $N = 20,000$.

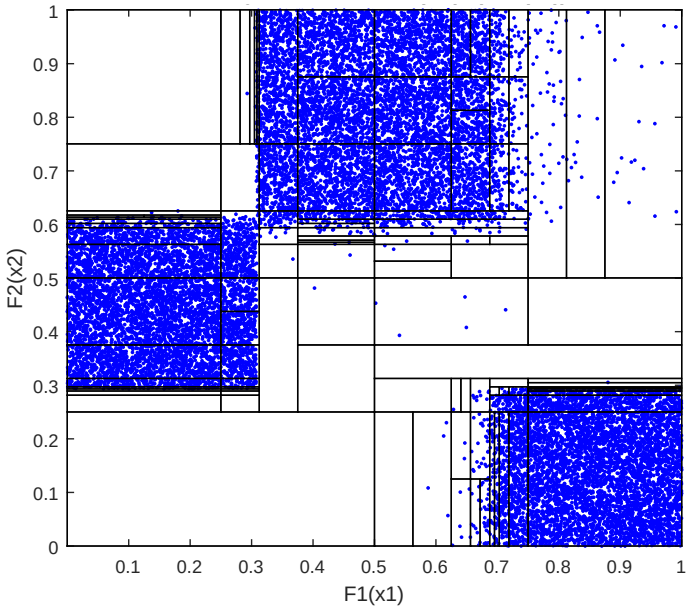


Figure 2.8: BSP cuts on copula-transformed sample space with $N = 20,000$ and $M = 200$. The number of BSP cuts is 87.

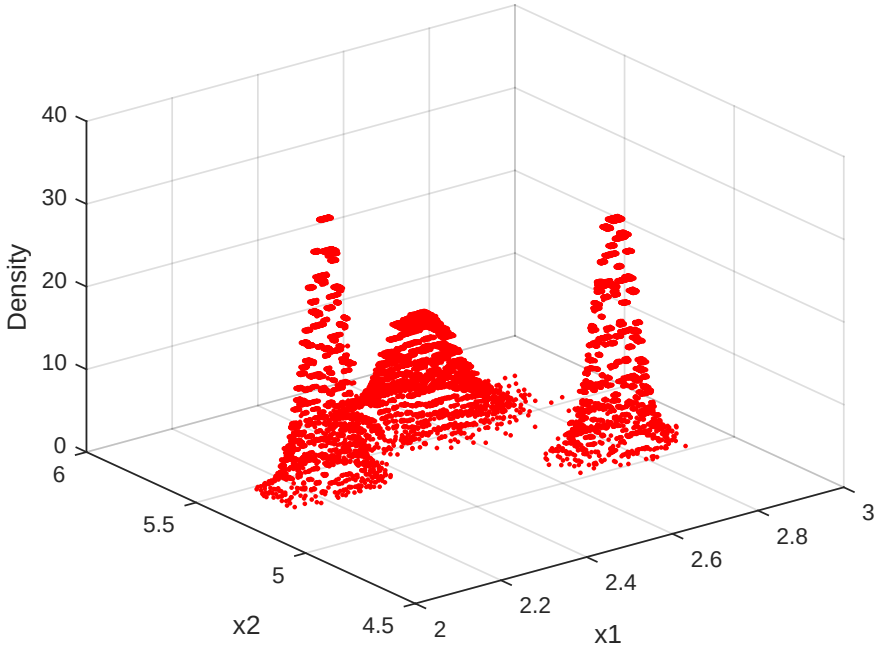


Figure 2.9: Estimated joint density, using copula transform with $N = 20,000$ and $M = 200$

Table 2.2

BSP with copula-transformed cuts on D-dimensional space, for three different options.

Option 1: $M = 200$ for marginals, $M = 200$ for copula, *Option 2*: $M = 1$ with MAP for marginals, $M = 200$ for copula, *Option 3*: $M = 1$ with MAP for marginals, $M = 1$ for copula.

		$D = 2$			$D = 32$			$D = 64$			$D = 128$			$D = 256$		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**
N=20k	Cuts	183	185	173	1694	1678	1625	3338	3262	3279	6666	6474	6485	13314	12914	12939
		(96+87)	(79+106)	(79+94)	(1614+80)	(1543+135)	(1517+108)	(3266+72)	(3152+110)	(3147+132)	(6569+97)	(6249+225)	(6302+183)	(13237+77)	(12649+265)	(12693+246)
	Time	19	13	0.1	278	58	1	552	87	2.6	842	197	10	1680	427	21
	KLD	0.018	0.024	0.033	0.19	0.93	1.33	0.31	1.62	2.02	1.51	4.22	3.90	1.59	6.84	6.59
	(std)	(0.0034)	(0.0116)	(0.0202)	(0.0727)	(0.4629)	(0.5576)	(0.0102)	(0.4665)	(0.7129)	(0.4599)	(0.7237)	(0.7129)	(0.3445)	(1.0132)	(0.7772)
N=100k	Cuts	293	294	284	2674	2654	2598	5277	5244	5153	10462	10400	10328	20837	20723	20621
		(156+137)	(152+142)	(150+134)	(2554+120)	(2526+128)	(2531+67)	(5155+122)	(5130+114)	(5115+38)	(10345+117)	(10277+123)	(10277+51)	(20734+103)	(20618+105)	(20618+3)
	Time	111	44	0.5	1262	110	5	2475	169	10	4254	293	50	8857	485	99
	KLD	0.007	0.008	0.01	0.068	0.077	0.75	0.13	0.14	0.88	0.31	0.39	1.10	0.52	0.57	1.56
	(std)	(0.0014)	(0.0018)	(0.00508)	(0.004)	(0.0049)	(0.4119)	(0.0044)	(0.0043)	(0.4499)	(0.1079)	(0.1224)	(0.4221)	(0.0068)	(0.0084)	(0.4499)

Figure 2.6 to Figure 2.9 illustrate BSP through the application of copula transform for the same $N = 20,000$ data samples generated by Eq. 2.3. Figure 2.6 presents the estimated marginal PDFs and CDFs from BSP. Discontinuities in the PDF plots correspond to cuts from BSP process. Figure 2.7 presents two random variables $F_1(X_1)$ and $F_2(X_2)$ in the transformed domain, which are in fact marginal CDFs of X_1 and X_2. Figure 2.8 shows the copula-transformed sample space and the cuts made by BSP process. For $N = 20,000$ and $M = 200$ the BSP has made a total of 183 cuts with 49 and 47 cuts for the two marginals X_1 and X_2, respectively. The number of cuts for estimating the copula-transformed density in the 2-dimensional space is 87, a significant reduction from 182 cuts in the direct method. However, as expected, and seen in Table 2.1, for a low dimensional problem of $d = 2$ the overhead associated with the copula transformation far outweighs the reduction in the number of cuts in 2-dimensional space, which results in a higher execution time. Further, notice the difference between the cuts in Figure 2.5 with a higher number of cuts in the high density areas, and Figure 2.9, where most cuts are made away from high density areas. As we will see later, the cuts in the high density areas involve much higher computational complexity. The estimated joint density from the BSP algorithm is shown in Figure 2.9. The KLD between the true density in Figure 2.5 and the estimated density in Figure 2.9 is only 0.018.

2.3.1 Copula Transform and Sample Partition Diversity

One important effect of copula transformation is that for each of the marginals, cuts are made in one-dimensional space, meaning that at each level j, there are only $j-1$ possible ways to make the next cut; alleviating the need for selecting a typically large value of M to improve density estimation through increased path diversity. Simulation results for the 2-dimensional example in Figure 2.6 for marginals and Figure 2.8 for copula-transformed partitions are shown in Figure 2.10 (a) and (b), respectively. As seen from Figure 2.10 (a) after a relatively small number of cuts, the partition scores for all $M = 200$ sample partitions merge towards a single value, indicating that all M independent paths eventually lead to the same or similar partitions. Plots in Figure 2.10 (a) also show that for two choices of $M = 1$, (Δ for MAP and $\square$ for random cut from PMF), the scores are very close to, (and for most cuts even better than), the best score obtained with $M = 200$. Thus, for all marginals, BSP can be applied with only one sample partition, $M = 1$ with MAP for all j cuts.

For the case of BSP in copula-transformed multidimensional space, in Figure 2.10 (b) despite the fact that most of the sample partitions tend to converge at relatively small number of cuts j, the algorithm needs to process a large number of sample partitions to maintain a good diversity. The scores for two cases of $M = 1$ are very close to the best score obtained with $M = 200$, but generally less than the best score obtained

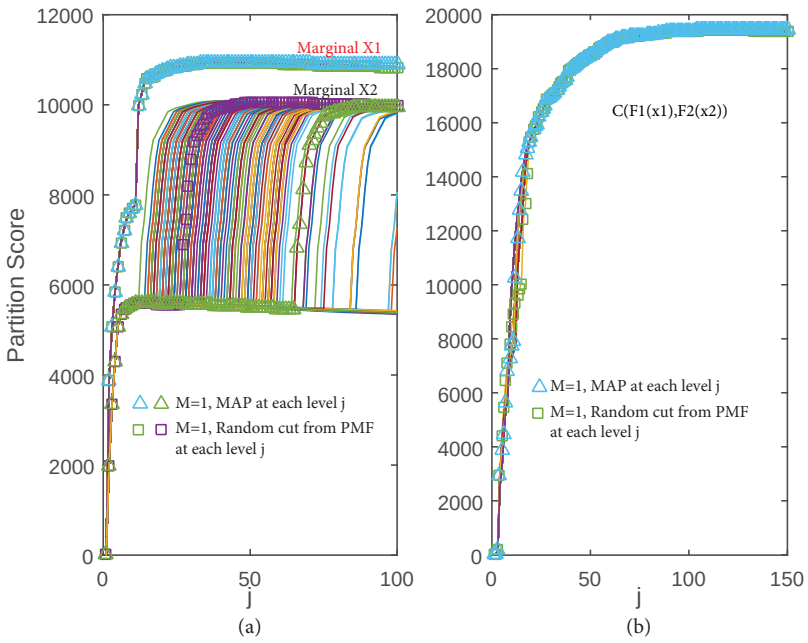


Figure 2.10: Partition scores with $N = 20,000$ and $M = 200$ sample partitions for examples for (a) marginals in Figure 2.6, (b) copula in Figure 2.8.

with $M = 200$.

To further investigate the effect of M on the density estimation, a set of simulations is performed with dimensions and distributions identical to Table 2.1, with two different dataset sizes. Simulation results are shown in Table 2.2, for a range of dimensions, from 2 to 256. In the basic setup (*Option 1*), M is 200 for all marginals as well as the copula-transformed density estimation part. At each level j, the location of the next subregion cut is decided based on a random draw from the conditional probability in Eq. 2.1. A variation of the basic set up is *Option 2*, where $M = 200$ for the copula-transformed part, but $M = 1$ for the marginals; i.e., for each of the marginals the algorithm only maintains one sample partition from beginning to the end of BSP.

At each level j, subregion p with the highest conditional probability s_{j_p} will be picked for the next cut (MAP estimate). Finally, in *Option 3*, $M = 1$ for all marginals as well as the copula-transformed estimation. In this setting, MAP estimate described above is used for the marginals only.

The values reported in this table are mean values obtained from multiple runs of the code: 32 runs for $D = 2$, 16 runs for $D = 32$, 8 runs for $D = 64$, and 4 runs for higher dimensions. The number of cuts are shown as the sum of marginal and copula cuts, with individual components presented as the pair in the parenthesis. The standard deviation corresponding to each KLD value is reported below it, in parenthesis.

As can be seen in Table 2.2, for small dimension of $D = 2$ the variations in the estimation error (KLD) is small across the three options, even when the dataset is relatively small. Therefore, M can be set to 1 for both marginals and the copula to gain one to two orders of magnitude reduction in the computational complexity. For larger dimensions (32 and 64), with $N = 20,000$, the degradation in KLD is significant and use of options 2 and 3 become less attractive. However, for large dataset of $N = 100,000$, with no appreciable increase in KLD, an order of magnitude reduction in computation time can be obtained by setting $M = 1$ for the marginals. A further one order magnitude reduction in computation time can be achieved, by setting $M = 1$ for both marginals and copula BSP, if an increase in KLD to more than 1.0 can be accepted.

The trend remains the same as the dimensions are increased to 128 and 256. It can

be seen that even for high-dimensional cases, the BSP method (with *Option 1* or *2*) is still able to provide good results, as long as the sample size is increased accordingly. In the following subsections, extended simulation results are presented, for a wider range of values of N, for the case of $D = 64$, to show how the estimation error is decreased by increasing the sample size.

Regarding computation time, it is observed that the time increases almost linearly with increasing dimensions. Computational complexity of the algorithm is discussed in further detail, in Sections 2.5 and 2.6.

2.4 Algorithm and Data Structures for BSP

2.4.1 Algorithm

When dealing with large, high-dimensional datasets, BSP method of density estimation can result in high computational complexity. The flowchart in Figure 2.11 illustrates my efficient implementation of the BSP through repeated evaluation of Eq. 2.1 and Eq. 2.2 in a loop. Similarly, the flowchart in Figure 2.12 illustrates the process of BSP using copula transformation, through evaluation of Eq. 2.4 and Eq. 2.5.

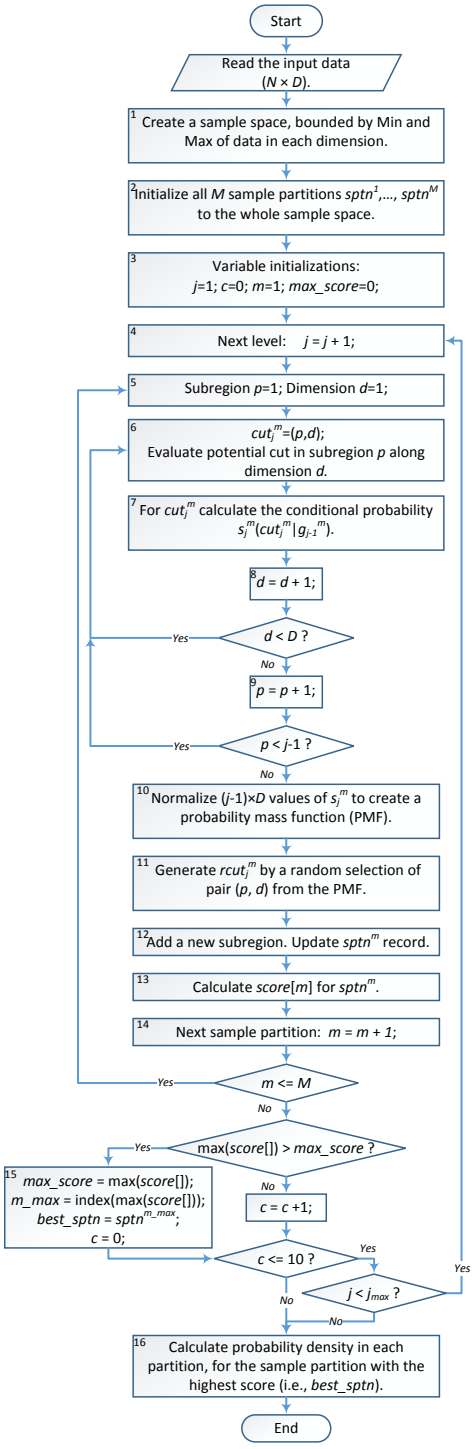


Figure 2.11: Flowchart for Bayesian sequential partitioning.

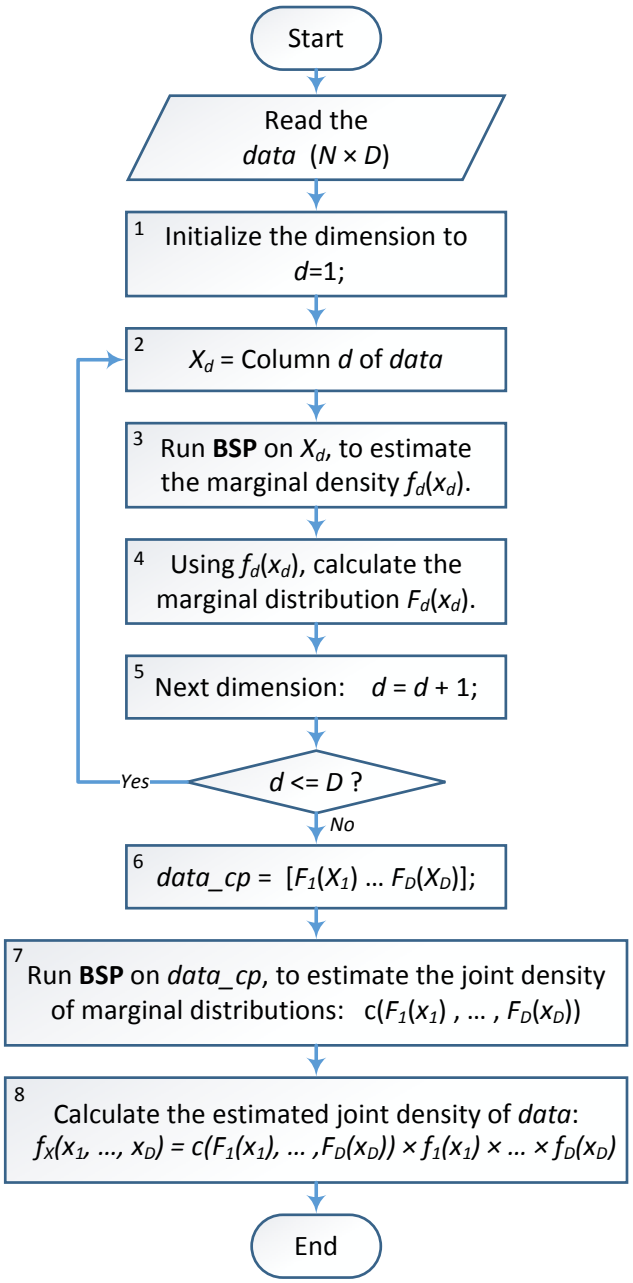


Figure 2.12: Flowchart for density estimation using copula transform.

2.4.2 Subregion Evaluation Reduction

To improve the speed of the algorithm in the flowchart of Figure 2.11, an important observation is made. With reference to Eq. 2.1, at each level j, there is no need to calculate $s_{j_{pd}}$ for all possible pd cuts. This is because the $s_{j_{pd}}$ values corresponding to all cuts, except the $2D$ potential cuts in the two recently modified subregions, are already calculated in the previous level, $j-1$. By properly storing and reusing these values, we can reduce the number of $s_{j_{pd}}$ evaluations from $(j-1)D$ to only $2D$; which is a huge reduction in computation time, when level j becomes large, in high-dimensional problems. This simplification eliminates the loop over p in the flowchart of Figure 2.11. Figure 2.2 illustrates this computational simplification in 2D sample space. At each level $j > 1$ there are only two possible cuts (marked in blue and red dashed lines) for each subregion, with a total of $2(j-1)$ possible cuts. The red dashed line identifies the location earmarked for the actual cut at level j. The resulting two subregions separated by a solid black line, due to a cut at level j are marked in beige. At each level $j \geq 3$, the algorithm only needs to calculate four new values of $s_{j_{pd}}$ that belong to these two new subregions. All remaining possible $s_{j_{pd}}$ values for cuts in subregions marked white are carried over from level $j-1$. In Figure 2.2, the left column shows all the possible cuts, with red lines marking the actual cut. This leads to the new partition shown in the right column. For example, at level $j = 5$, there are $(5-1) \times 2 = 8$ possible ways to make the next cut. But the

probabilities associated with the possible cuts in subregions 1 and 3 (cut numbers 1, 2, 5, and 6) have already been calculated in the previous level $j = 4$, and are available for reuse. So the algorithm only needs to evaluate the $s_{j_{pd}}$ values for cut numbers 3, 4, 7, and 8.

2.4.3 Data Structures

Main data structures for implementing the described BSP algorithm are listed in Table 2.3 [59]. The data structures are designed for efficient memory access and minimum data movement. They are also designed for the ease of mapping into an efficient parallel processing paradigms of OPENMP® and MPI. All data structures are organized for optimal column-major memory access, the method of choice in MATLAB®. Some of the data structures listed in Table 2.3 are for the purpose of estimation of one-dimensional marginal densities, while some others are exclusively designed for copula density estimation, and some structures are common for both estimations.

At each level j, subsequent to a cut in one of the existing subregions, the data points in each of the two new subregions need to be identified and their total number counted and stored in the structure `nSub`. This is the most time consuming part of BSP algorithm for large and high-dimensional datasets. To reduce the computational complexity associated with this operation, the $N \times D$ input dataset structure, `data`, is augmented with an additional column p to store the corresponding subregion number

for each of the data points. Then the whole structure is rearranged (partially sorted) such that all the points with the same subregion identity are stored in a contiguous block. The implemented scheme is shown in Figure 2.13. A pointer structure, `dataDistLimits`, stores the indices for the first (marked green) and the last element (marked red) in each subregion. The efficiency of the proposed structure comes from the fact that when at level j, subregion t is cut, the algorithm only needs to sort the subset of the `data` structure marked as t. This partitioning scheme of dataset structure, `data` works well for small dimensions less than 5. However, for larger dimensions, even the limited subregion sorting process requires movement of a large amount of multi-dimensional data. To reduce the data movement for dimensions larger than five, the structure shown in Figure 2.14 (a) is adopted where an intermediate data structure `dataDist` stores the indices to `data`. This way, the subregion sorting process involves a much simpler 2-column structure, instead of a D-column structure. In the optimized implementation of BSP using copula transform, for each of the marginals, a simpler structure, shown in Figure 2.14 (b), is used. Before processing each marginal d, its corresponding column is copied from `data` to `dataMarginal`. After processing, `dataMarginal` will contain the CDF for that marginal, which is copied back to the corresponding column of `data` for copula processing, using the structure in Figure 2.14 (a).

Table 2.3
List of main data structures used in copula-based implementation of the BSP algorithm.

Name	**Dimensions**	**Description**
`data`	$N \times D$	Contains the whole input dataset or copula-transformed data, i.e. N rows of D-dimensional data.
`dataMarginal`	$N \times 3 \times M$	For each of the M sample partitions, `dataMarginal` stores distribution of the marginal data for dimension d in partition subregions. The first column contains an index to the original dataset stored in `data`. The second column contains a copy of column d of `data`. The third column stores the corresponding subregion numbers for the data from column 2. After processing of each marginal d, the second column of `dataMarginal` will contain its marginal CDF, and is copied back to the corresponding column of `data` for copula processing.

Continued on next page

Table 2.3 – *Continued from previous page*

Name	**Dimensions**	**Description**
`dataDist`	$N \times 2 \times M$	For each of the M sample partitions, `dataDist` stores distribution of the data in the partition subregions. The first column contains an index to the original dataset stored in `data`, and the second column stores the subregion numbers corresponding to the indices from column 1.
`dataDistLimits`	$2 \times j_{max} \times M$	Maintains pointers to the start and end entries of each subregion in `dataDist`
`subCoords`	$2D \times j_{max} \times M$	Stores coordinates of the subregions. Each subregion is specified by its lower and higher boundaries in each dimension.
`sjLog`	$D \times j_{max} \times M$	Stores the calculated probabilities for cutting each of the existing subregions along any of dimensions, at each level j. All values are in *log* domain.
`sj`	$D \times j_{max}$	Stores the *log* normalized probabilities from `sjLog`.
`scores`	$M \times 1$	Stores the scores for sample partitions.

Continued on next page

Table 2.3 – *Continued from previous page*

Name	**Dimensions**	**Description**
`nSub`	$j_{max} \times M$	Stores the number of data points in each of the subregions for each of M sample partitions.
`nSubMAP`	$j_{max} \times 1$	Records the state of one column of `nSub` with the highest partition score, when the BSP meets the stopping criteria.
`vSub`	$j_{max} \times M$	Stores the volume of the subregions for each of the M sample partitions.
`vSubMAP`	$j_{max} \times 1$	Records the state of one column of `vSub` with the highest partition score, when the BSP meets the stopping criteria.
`bestDistMAP`	$N \times 2$	Stores a subset of `dataDist` structure for the sample partition with the highest partition score.
`rCuts`	$M \times 1$	Stores the subregion number that is randomly cut at each level j.

Continued on next page

Table 2.3 – *Continued from previous page*

Name	**Dimensions**	**Description**
`CoordSubMarMAPs`	$2 \times j_{max} \times D$	Each columns of this structure corresponds to one of the marginal densities and stores the final coordinates of the subregions for the marginal sample partition with the highest partition score.
`densSubMarMAPs`	$j_{max} \times D$	Each columns of this structure corresponds to one of the marginal densities and stores the final estimated subregion marginal densities.
`CoordCopulaMAP`	$2D \times j_{max}$	Stores the final coordinates of the subregions, taken from the partition with the highest partition score.
`densCopulaMAP`	$j_{max} \times 1$	Stores the final values of the joint probability densities, for the copula-transformed density estimation.

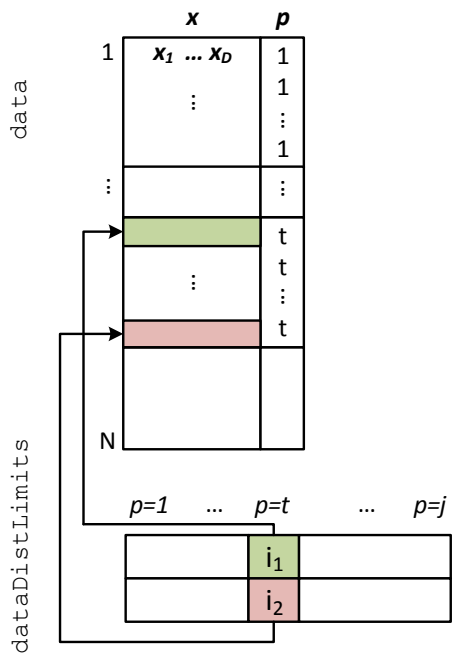


Figure 2.13: Data structure extended, to store distribution of the data points in subregions.

2.5 Density Estimation Simulation Results

This section presents the results obtained from MATLAB® simulation runs on a single Intel® Core i7-3820, 3.60 GHz, with 32 GB RAM, for the performance evaluations of the algorithm.

Density estimation is performed for the range of sample data sizes from $N = 10,000$ up to $N = 1,000,000$, and range of dimensions, from $D = 2$ up to $D = 64$. The simulations aim at examining the impact of different parameters on computational efficiency, measured in execution time, and estimation accuracy measured as KLD. Figure 2.15 presents the execution time and KLD for a 64-dimensional dataset versus

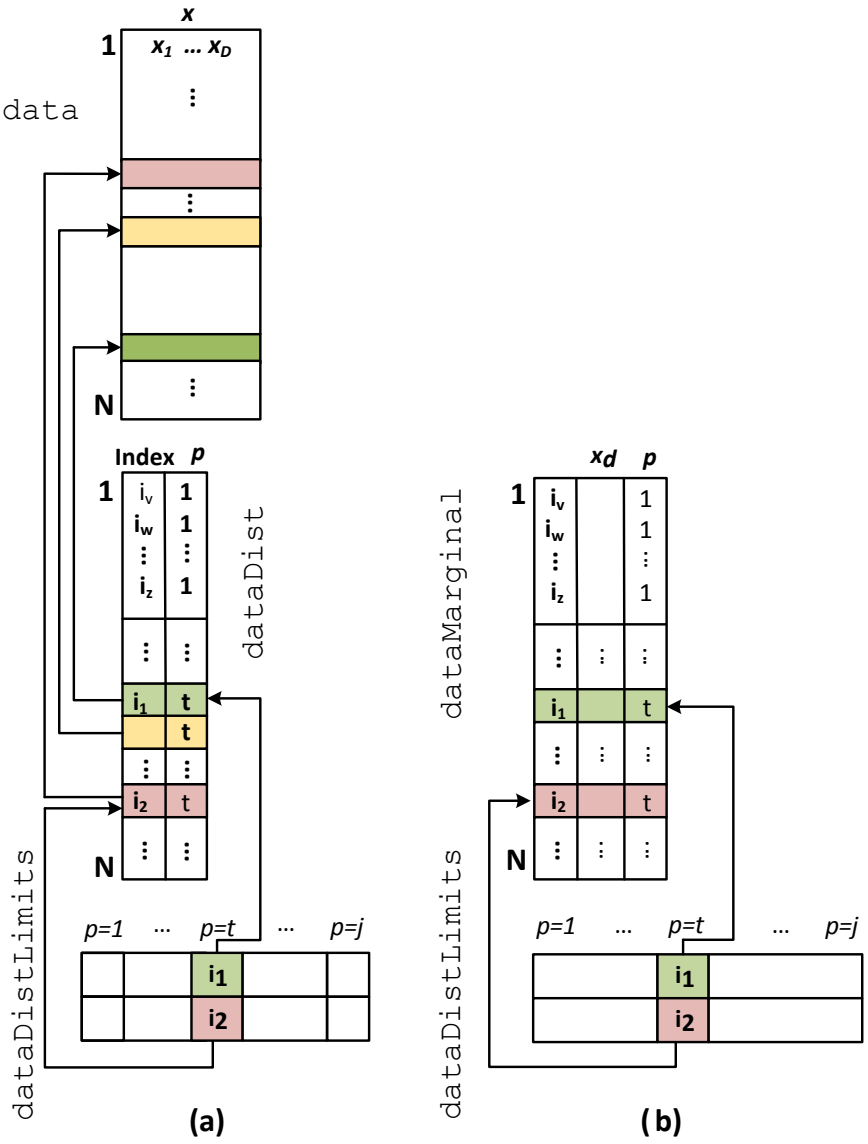


Figure 2.14: Optimized data structure for storing distribution of data points in subregions for (a) copula-transformed estimation where intermediate data structure `dataDist` stores the indices to `data`, and (b) for each of the marginals, where the intermediate step is not required.

the sample sizes N, with the number of sample partitions M as a parameter. Clearly, a larger sample dataset improves accuracy, at the cost of increased execution time. It can be seen that the execution time grows almost linearly with the sample size N.

The KLD between the actual density and the estimated density, on the other hand, decreases exponentially with sample size. For this particular example, increasing the sample size over $N = 300,000$ has little impact in improving the estimation accuracy. While the execution time increases linearly with M, the KLD exhibits small sensitivity to M. It is also seen that the choice of $M = 1$ for the marginals and $M = 200$ for the copula transformation results in no appreciable increase in KLD, with a six-fold reduction in computational complexity, when compared with the next closest case of $M = 50$ for both marginals and copula. A further 12-fold improvement in the execution time can be seen for the choice of $M = 1$ for both marginals and the copula transformation. However, KLD on average remains higher than the case of $M = 1$ for the marginals and $M = 200$ for the copula transformation, by about a little more than 1.0. It also can be seen that for this case, due to the lack of diversity in the selection of the sample partitions, the KLD is relatively unstable across the range of N values.

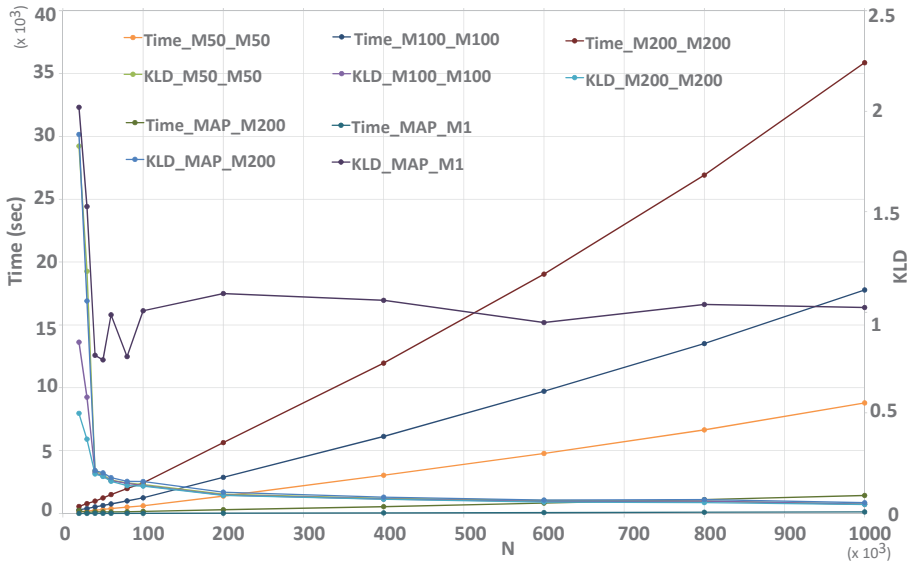


Figure 2.15: Execution time and KLD vs. sample size (N) for different values of M for a 64-D dataset.

2.6 Complexity Analysis and Parallelization

2.6.1 Complexity Analysis

The profiling of the algorithm presented in the flowcharts of Figure 2.11 and Figure 2.12 for *Option 1*, ($M = 200$ for marginals, $M = 200$ for copula), with $N = 100,000$ and $D = 64$ in Table 2.2 reveals that only 5% of the execution time is spent on the copula part. One reason is that only 2% of 5277 cuts are attributed to the copula. However, since a copula cut goes over all the 64 dimensions it is expected to have a much higher complexity (close to 64 times higher) compared to the complexity of a

marginal cut that goes over a single dimension; but the profiling results reveal that the average computational complexity for a copula cut is only about 2 times larger than that of the marginal cuts! There are two reasons for this ratio to be low. First, referring to the flowchart in Figure 2.11, it is noted that steps 10 to 14 in the flowchart are outside the loop for d, and therefore, executed only once for each iteration of m irrespective of the value of d. The profiling results further reveal that over 81% of the overall execution time is spent on the execution of these steps, with step 12 being the overwhelming contributor (over 78%). From the number of cuts in Table 2.2, it can be observed that on average every round of execution of steps 10 to 14 for a copula cut, there are 40 execution rounds of the same steps for a marginal cut. However, the overall complexity of one round of execution of these steps (in most part due to step 12) is more than three times less for a copula cut compared to a marginal cut. This is because in the copula case the fraction of cuts in the high density regions requiring time consuming count and sort operations across a large number of data points, and update of data structures in Fig 2.13 and 2.14 is far less than that for the marginals. This can be observed from Figure 2.8, where large blocks of high density areas in blue have no cuts. Further, for by the same token, the complexity of one round of execution of steps in the d loop, step 6 (involving the count of data points in each newly created subregion) to 9 is much less for a copula cut than for a marginal cut. From the preceding discussion, due to the fact that copula cuts form only a small fraction of the overall time, attempts to parallelize this part of the algorithm yields

no benefit. Therefore, the work presented later in Subsection 2.6.3 only focuses on parallelization over the marginals.

2.6.2 Effect of covariance matrix on performance of BSP

For illustration simplicity, in Section 2.2, for the Gaussian mixture distribution in Eq. 2.3, it was assumed that $\Sigma_r[i,j] = 0$ for $i \neq j$. This is equivalent to $R_r[i,j] = 0$ for $i = j$ and $R_r[i,i] = 1$, where $R_r = (\Sigma_r)^{-\frac{1}{2}}\Sigma_r(\Sigma_r)^{-\frac{1}{2}}$ is the correlation matrix. In general, however, the correlation between marginals can have non-zero values. In order to investigate the effect of correlation matrix on performance of BSP, Table 2.4 presents the KLD and execution time performance of BSP for a range of correlation coefficients between 0 and 1, for the same 64-dimensional example studied before. From the data in the table, the KLD values remain small for correlation coefficients as high as 0.95. The contribution of copula to the total computation time increases linearly from about 6% to about 20% for the change in the correlation coefficient from 0 to 0.95. The increase in the number of cuts follows the same trend.

It should be noted that in the multivariate analysis, where there is high correlation between the marginals the method of principal component analysis (PCA) [60] can be used to transform the dataset into a new set of variables in smaller number of dimensions (principal components) that are uncorrelated. The method of BSP can then be applied to the transformed dataset.

Table 2.4
Impact of correlation on computation time and estimation accuracy ($N = 100,000$, $D = 64$).

	Time			Cuts			KLD
R	**Marginal**	**Copula**	**Total**	**Marginal**	**Copula**	**Total**	
0	2315 (93.7%)	160 (6.3%)	2475	5155	122	5277	0.13
0.2	2288 (92.8%)	178 (7.2%)	2466	5106	178	5284	0.13
0.4	2282 (90.5%)	239 (9.5%)	2521	5078	246	5324	0.14
0.6	2282 (89.9%)	256 (10.1%)	2538	5133	269	5402	0.23
0.8	2280 (85.0%)	402 (15.0%)	2682	5133	464	5597	0.15
0.9	2352 (82.6%)	494 (17.4%)	2846	5133	579	5712	0.16
0.95	2302 (80.0%)	574 (20.0%)	2876	5133	671	5804	0.18
1.0	2322 (25.1%)	6930 (74.9%)	9252	5148	4177	9325	14.71

2.6.3 Parallelization

The proposed algorithm in the flowcharts of Figure 2.11, and the data structures in Table 2.3, Figure 2.13 and Figure 2.14, have been designed suitably for the ease of parallelization around BSP parameters; dataset elements $1 \leq n \leq N$, sample partitions $1 \leq m \leq M$, dimensions $1 \leq d \leq D$ and subregions $1 \leq p \leq j-1$. In this section, the discussion is limited to the coarse-grain parallelization around m and d. Fine-grain parallelization over data sample n is best achieved through massively parallel architectures of graphical processing unit (GPU) and is limited to

the counting and sorting of data samples for the purpose of density estimation in the subregions. Due to significant overhead of data transfer between the host CPU and the device GPU, and the non-coalesced memory access pattern of the data structures in Figure 2.14, it is not possible to take full advantage of the GPU computing fabric to efficiently accelerate the counting and sorting operations on the GPU. Following the discussion in Section 2.4 and with reference to Figure 2.2, parallelization over p is no more needed due to elimination of the loop over p due to simplification presented in Section 2.4.2. Therefore, this work has focused on the coarse-grain parallelization over d and m, where it is expected to obtain the maximum speedup gains. The chosen platform for parallelization is a four-core CPU (Intel i7-3820, 3.60 GHz, with 32 GB RAM).

2.6.3.1 Parallelization over d

For the BSP over the marginals, since the dimensions are decoupled from each other, it is expected to gain the maximum benefit in parallelization over d. The OPENMP®[61] programming model ("`parfor`" is used in MATLAB®) for parallelization over d. The reason for this choice is that iteration over the marginals are completely independent of each other and there is no exchange of data between the parallel execution threads. With a 4-core CPU, a four-way parallelization can be achieved for a 64-dimensional problem, with each worker core being allocated

the workload for 16 marginals. Unfortunately, due to uneven workload across the marginals, the overhead of setting up the parallel streams, and non-parallelized execution portion for copula cuts, the maximum achievable speedup, as seen form Table 2.5, is no more than 3.3 for *Option 1* ($M = 200$ for marginals, $M = 200$ for copula) and *Option 3* (MAP for marginals, $M = 1$ for copula), with $N = 100,000$ and $D = 64$. Table 2.5 also presents the speedup for *Option 2* (MAP for marginals, $M = 200$ for copula). The low speedup factor of 1.4 for *Option 2* is in keeping with the fact that with $M = 1$, the execution time for the marginals has reduced by a large factor and is of the similar order to the non-parallelized copula.

2.6.3.2 Parallelization over m

In parallelization over m, in each level j, evaluations of M sample partitions are only partially independent of each other. The decision diamond in the flowchart of Figure 2.11, where `max(score[])` is evaluated outside the parallel loop, forms the synchronization barrier for the computing worker cores. MPI [62] programming model is used for parallelization over m. The reason for this choice is that iterations of m are only partially independent of each other and there is a need for the exchange of data between the parallel execution threads through the MPI message passing constructs ("`spmd`" in MATLAB®). Considering that step 15 in the flowchart forms about 10% of the execution time, and the significant overhead of synchronization in the diamond

Table 2.5
Speedup rates for parallelization over d and m, on a 4-core machine ($N = 100,000$, $D = 64$).

Scenario	Speedup		
	Option 1	**Option 2**	**Option 3**
Parallelization over d	3.3	1.4	3.3
Parallelization over m	2.2	-	-

decision box, overhead of distribution of data among the workers in step 15, and initial overhead of distribution of large data structures across the workers, plus the overhead of non-prallellized copula cuts, the maximum speedup for four workers on a 4-core machine is no more than 2.2 as seen form Table 2.5.

2.7 Use of alternate methods for the marginal densities

2.7.1 Marginals density estimation with the KDE method

With the separation of high dimensional density estimation into one-dimensional and copula parts we are able to perform the density estimation for the marginals using the KDE method in Eq. 1.2. However, unlike the method of BSP, the choice the bandwidth h and kernel function in Eq. 1.2 becomes critical in the accurate estimation of the density. The Gaussian kernel function and rule of thumb bandwidth estimator

of $h = 1.06\sigma n^{-\frac{1}{5}}$ [19] are used in this work. Table 2.6 compares the density estimation results, using the BSP and KDE for the same 64-dimensional dataset used before. Three observations can be made from the data in the table. First, for the BSP, the KLD improves by a factor of up to 18.20 (Option 2) and the execution time increases by up to 22 (Option 1) when the size of dataset increases by a factor of 20, from $20k$ to $400k$. For the KDE, the improvement in KLD is no more than 1.61 (nGrid=1000), while the increase in the execution time goes by a factor of up to 64 (nGrid=250). The second observation is that the BSP is a more flexible technique. The execution times and the KLD values for the BSP change by up to two orders magnitude for the three BSP options. This is not the case, however, for the KDE where the changes in both execution times and the KLD values are much less significant. The third observation is the fact that for small data size of $N = 20k$, similar KLD and execution times can be obtained for the BSP and KDE techniques. However, KDE loses to BSP in the execution time by two orders magnitude for a similar KLD performance. For example, BSP Option 1 is 136 times faster than KDE with nGrid of 500 for the similar KLD values.

To further evaluate the relative performance of BSP and KDE, a 64-dimensional dataset with a skewed distribution is examined. For the first three marginals, the same distribution as the previous dataset is maintained. Dimensions 4 to 64, however,

come from the following mixture of Beta distribution [56]:

$$X \sim 0.6 \times B(2, 8) + 0.4 \times B(120, 14)$$

It is a bimodal distribution, with the $B(2, 8)$ having rather a wide PDF with a positive skew, and the $B(120, 14)$ mode having a much sharper and almost symmetric PDF. Table 2.7 presents the results. Comparing the results with those in Table 2.6, the execution times have gone up for the KDE case due to higher number of cuts in copula distribution. For BSP, being a more adaptable method, the changes in the number of cuts have been by much smaller margins. However, while the KLD values for the BSP have remained unchanged, they have deteriorated for the KDE by an order of magnitude.

Table 2.6
Comparison of density estimations using the methods of BSP, KDE and median-based cuts for the marginals ($D = 64$).

For KDE, $M = 200$ has been set for the copula part. Results are average of 10 runs. (5 runs for $N = 400k$)

		BSP			Median			KDE		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**nGrid=250**	**nGrid=500**	**nGrid=1000**
N=20k	Cuts	3338	3262	3279	3194	2196	3021	16086	32094	64124
		(3266+72)	(3152+110)	(3147+132)	(3085+109)	(2076+120)	(3019+2)	(16000+86)	(32000+94)	(64000+124)
	Time	397	43	5	361	75	5	86	104	74
	KLD	0.37	0.42	1.24	0.39	1.74	1.45	0.99	0.72	0.61
	(std)	(0.0092)	(0.0056)	(0.33)	(0.018)	(0.073)	(0.099)	(0.68)	(0.54)	(0.32)
N=100k	Cuts	5277	5244	5153	5065	4268	4895	16165	32172	64150
		(5155+122)	(5130+114)	(5115+38)	(4887+178)	(4088+180)	(4831+64)	(16000+165)	(32000+172)	(64000+150)
	Time	1770	147	25	1621	204	24	645	660	500
	KLD	0.15	0.18	1.01	0.15	0.22	1.02	0.49	0.38	0.45
	(std)	(0.0058)	(0.0043)	(0.45)	(0.01)	(0.042)	(0.20)	(0.46)	(0.44)	(0.32)
N=200k	Cuts	6606	6496	6356	6151	6807	5909	16209	32198	64189
		(6447+159)	(6346+150)	(6281+75)	(5934+217)	(5822+5)	(5863+46)	(16000+209)	(32000+198)	(64000+189)
	Time	3846	281	21	3311	371	48	1805	1428	1469
	KLD	0.10	0.12	1.03	0.098	0.10	1.07	0.71	0.42	0.35
	(std)	(0.0047)	(0.0031)	(0.16)	(0.0024)	(0.0027)	(0.15)	(0.80)	(0.47)	(0.37)
N=400k	Cuts	7920	7881	7756	7610	7415	7169	16246	32255	64279
		(7736+184)	(7683+198)	(7683+73)	(7272+263)	(7146+269)	(7024+45)	(16000+246)	(32000+255)	(64000+279)
	Time	8495	531	42	7157	671	98	5582	5727	4719
	KLD	0.067	0.083	1.02	0.065	0.067	1.04	0.87	0.66	0.63
	(std)	(0.0055)	(0.0011)	(0.11)	(0.002)	(0.0017)	(0.014)	(0.63)	(0.12)	(0.19)

2.7.2 Marginals density estimation with median-based cuts

In an attempt to make the sequential cuts a better fit to the data density pattern, this section explores the idea of replacing the previously described mid-point binary cuts with *median-based* cuts. As explained in Section 2.2, in the original BSP method, sequential cuts are made in the sample space using the mid-point BP scheme, i.e., at each level, one of the existing subregions is cut into two equal halves. So, the cut location is always at the center point of the selected subregion, regardless of distribution of data in that subregion. In median-based approach, on the other hand, the location of the cut is at the median point of the data samples in that subregion. The decision on which subregion to cut at level j is made using a conditional probability, similar to Eq. 2.1, with the difference that with median-based cuts, the volumes of the subregions, v_p, $v_{pd}^{(1)}$ and $v_{pd}^{(2)}$ also appear in the equation:

$$s_{j_{pd}}(cut_{j_{pd}}|g_{j-1}) = C_{j-1}\Big(\frac{\Gamma(n_{pd}^{(1)})\Gamma(n_{pd}^{(2)})}{\Gamma(n_p)}\Big)\Big(\frac{v_p}{v_{pd}^{(1)}v_{pd}^{(2)}}\Big)^{n_p} \tag{2.6}$$

Also, the partition score calculation will be slightly different from Eq. 2.2.

This idea has been tried for the 64-dimensional data described before. As the results presented in Tables 2.6 and 2.7 show, for all 3 options, and for all different values of N covered in this simulation, using the median-based method for estimating the marginals reduces the number of marginal cuts, compared to the original method

of mid-point binary cuts. This is because firstly, the median-based approach tries to use the information about the distribution of data in the subregions to decide which subregion to cut; and secondly, once the subregion to cut is picked, it makes the cut in a more data-adaptive fashion. For *Option 1*, where the marginal cuts are the dominant part of the computation time, using the median-based cuts method for marginals leads into saving the overall computation time. Regarding estimation accuracy, both methods have similar performance in almost all cases presented in the tables. While the number of cuts for the method of the median is always smaller than the mid-point method, and both methods exhibit similar measures of accuracy, the computationally complexity, except for *Option 1*, is not better. That is because searching for the median points incurs significant computational overhead. Therefore, the rest of this work continues to use the mid-point binary cuts method for further discussions, simulations and analyses.

Table 2.7

Comparison of density estimations using the methods of BSP, KDE and median-based cuts, for the marginals for a bimodal skewed Beta distribution ($D = 64$).

For KDE, $M = 200$ has been set for the copula part. Dimensions 4 to 64 each have a bimodal Beta distribution.

		BSP			Median			KDE		
		Option 1	**Option 2**	**Option 3**	**Option 1**	**Option 2**	**Option 3**	**nGrid=250**	**nGrid=500**	**nGrid=1000**
N=20k	Cuts	2825	2804	2731	2725	2316	2087	16114	32096	64142
		(2735+90)	(2729+75)	(2729+2)	(2616+109)	(2196+120)	(2081+6)	(16000+114)	(32000+96)	(64000+142)
	Time	357	40	5	312	75	5	32	127	172
	KLD	0.35	0.36	1.20	0.65	1.74	2.84	7.7	7.6	7.6
	(std)	(0.0043)	(0.0073)	(0.30)	(0.08)	(0.073)	(0.084)	(0.61)	(0.29)	(0.24)
N=100k	Cuts	4526	4520	4399	4414	4448	4102	16229	32201	64223
		(4405+121)	(4397+123)	(4397+2)	(4236+178)	(4268+180)	(4095+7)	(16000+229)	(32000+201)	(64000+223)
	Time	1641	149	25	1531	204	23	722	741	805
	KLD	0.13	0.14	1.10	0.14	0.22	1.22	4.9	4.8	4.8
	(std)	(0.0023)	(0.0021)	(0.15)	(0.008)	(0.042)	(0.11)	(0.74)	(0.38)	(0.27)
N=200k	Cuts	5633	5597	5461	5348	5209	5007	16230	32221	64229
		(5490+143)	(5449+148)	(5450+11)	(5138+210)	(4988+221)	(4988+19)	(16000+230)	(32000+221)	(64000+229)
	Time	3460	280	51	3070	364	47	1478	1499	1577
	KLD	0.0867	0.092	1.06	0.091	0.11	1.13	3.78	3.73	3.78
	(std)	(0.0036)	(0.0013)	(0.039)	(0.0018)	(0.009)	(0.11)	(0.67)	(0.60)	(0.51)
N=400k	Cuts	7349	6990	6805	6522	6331	6150	16274	32283	64261
		(7169+180)	(6802+188)	(6802+3)	(6263+259)	(6073+258)	(6072+78)	(16000+274)	(32000+283)	(64000+261)
	Time	7764	547	100	6625	661	97	4716	4746	4848
	KLD	0.054	0.060	1.04	0.067	0.078	0.91	3.05	2.72	2.94
	(std)	(0.0044)	(0.0008)	(0.0009)	(0.0064)	(0.005)	(0.22)	(1.00)	(0.80)	(0.39)

2.8 Density-based classification and clustering

In this section it is shown how the multivariate density estimation framework developed in this chapter can be used as a basis for other data mining techniques. Specifically, some cases of density-based classification and clustering are presented, as examples of such applications.

2.8.1 Density-based classification

Classification [63] [64] is a supervised learning algorithm, which aims at assigning new observations to one of several mutually exclusive categories. The categorization of new unlabeled observations is performed on the basis of a set of observed data (training set) with known category memberships. Some well-known classification methods include Classification Trees [65] [66], K-Nearest Neighbors (KNN) [67] [68], Support Vector Machine (SVM) [69] [70] [71], and Linear Discriminant Analysis (LDA) [72] [73].

Density-based classification methods use the results obtained from a density estimator to extract classification rules [3] [15] [64]. In this section, BSP is used as the density estimator in a density-based classification framework.

In performing classification for a sample dataset consisting of L different classes

C_l $(l = 1 \cdots L)$, the first step is to divide the dataset into a training set and a test set. For the training set, BSP is used to separately estimate the density $\hat{f}_l$ for the data in each class l. Then, each new unclassified sample datum from test set can be assigned to one of the classes $C_1 \cdots C_L$, using the Bayes classifier [74] [75] [64], by utilizing the density estimation results for all classes,

$$\hat{C}(x) = \underset{l}{\operatorname{argmax}} \hat{f}(C_l|x) = \underset{l}{\operatorname{argmax}} \hat{f}(x|C_l) \times \hat{f}(C_l) \tag{2.7}$$

where $\hat{f}(x|C_l)$ is obtained from class density estimation results and $\hat{f}(C_l)$ from the prior knowledge of how classes are assigned to the labeled data in training set.

Table 2.8 compares the classification rates obtained from three BSP options with the results from some other widely used classification methods, for some example problems. The first example is the synthetic trimodal 2-dimensional dataset used previously in demonstration of BSP algorithm (Figure 2.3). Each of the three normal distributions correspond to a distinct class. As can be seen, the data is composed of three distinct clusters and thus it is expected to see very high classification rates. It is also observed that the difference between classification rates for all the three BSP options is negligible. Further, the change in the classification rates with the size of dataset is not appreciable.

The second example involves the classification of the Gamma telescope dataset,

Table 2.8
Classification rates for some sample datasets

Dataset	BSP			KNN	LDA	Classification
	Opt. 1	Opt. 2	Opt. 3	(k=5)		Tree
Synthetic 2D (N=20k)	98.27%	98.42%	98.29%	100%	99.96%	99.39%
Synthetic 2D (N=100k)	98.19%	98.24%	98.22%	100%	99.90%	99.22%
MAGIC	76.93%	76.57%	76.43%	79.28%	77.55%	82.49%

MAGIC [76] [77]. The dataset contains 19,020 observations of 10-dimensional data divided between two classes: 12,332 instances of *gamma* and 6,688 instances of *hadron*. As can be seen in Table 2.8, BSP-based classification works almost as well as the other two techniques.

2.8.2 Density-based clustering

Cluster analysis [41] [78] [79] [80] is an unsupervised learning algorithm that aims at classifying the unlabeled data into several categories, based on similarities (or differences). There are generally two main approaches to data clustering: hierarchical and partitional. Hierarchical techniques [81] either start with a single large cluster and try to split it (agglomerative techniques), or start with taking each single point as a separate cluster and try to merge them, based on some distance measure, to form clusters (divisive methods) [82]. In contrast, in partitional clustering methods, if it is desired to have K clusters, all those K clusters are found at the same time [82].

Numerous hierarchical and partitional algorithms have been developed for clustering

[83]. Many of these methods deploy some measure of distance, such as the Euclidean distance, as the basis for evaluating the similarity between each pair of observations.

K-means [84] is probably one of the most popular partitional clustering methods. It tries to find K cluster *centroids*, each identifying one cluster, and then assign samples to the clusters based on a nearest neighbour algorithm. In order to divide the data into K separate clusters, the algorithm starts by randomly selecting K points as the initial centroids; assigns points to clusters, based on distances from the centroids; then it computes the mean of each cluster to update the locations of the cluster centroids. These steps will be repeated until the algorithm reaches convergence, i.e., the cluster centroids do not change any more. In K-means method, the desired number of clusters K needs to be provided to the algorithm. Furthermore, since each data point is always assigned to its nearest cluster center, this method does not perform well in detecting non-spherical clusters.

In density-based clustering algorithms [85] [86] [87] [88], each cluster is identified as a high-density region surrounded by some lower density regions that separate it from other clusters. In the density-based clustering algorithm proposed by [89], a cluster center is defined as a point with high local density, which is surrounded by lower density points, and is located at a relatively large distance from any point with a higher local density. The algorithm needs the pairwise distances between all the points. For data point i, it estimates the local density ρ_i by counting the number of

points located within a cutoff distance d_c from point i. That is,

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{2.8}$$

where $\chi(x) = 1$ if $x < 0$ and $\chi(x) = 0$, otherwise.

Also, for each point i, the parameter δ_i is defined as the distance between point i and the closest point with a higher local density, i.e.,

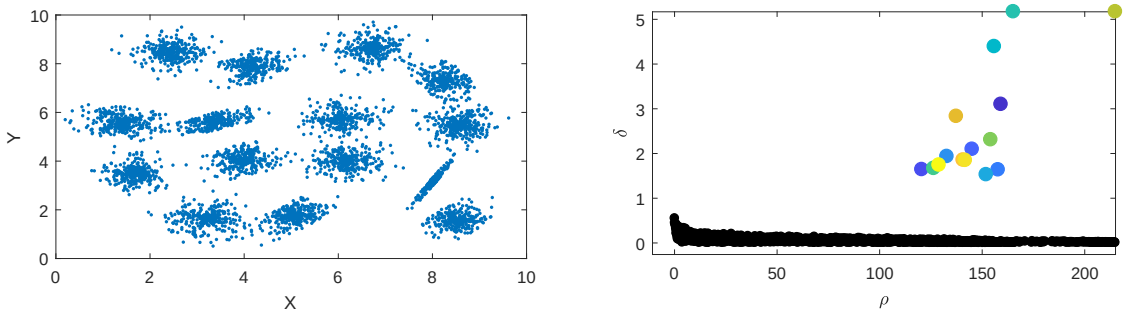
$$\delta_i = min(d_{ij}) \quad , \quad j : \rho_j > \rho_i \tag{2.9}$$

For the point p with highest density, the value of δ is set to the maximum distance, i.e., $\delta_p = max_j(d_{pj})$, p: $\rho_p = max(\rho)$.

As a rule of thumb provided in [89], the cutoff distance d_c can be chosen so that the average number of neighbors is around 1 to 2% of the total number of points in the dataset. Once ρ_i and δ_i are computed for all data points $i = 1 \cdots N$, the cluster cores are identified, and then each point is assigned to the same cluster as its nearest neighbor of higher density.

Figure 2.16 shows a 2D example with 15 clusters of various shapes [90]. The plot on the left is the original 2D data and the plot on the right shows the decision graph created by plotting δ vs ρ. It shows how the cluster centers stand out, as higher density points with large values of δ. The 15 cluster centers are identified and cluster

assignation is performed, as shown in Figure 2.17. The crosses show the locations of the identified cluster centers.



(a) Original unlabeled data

(b) Clustering decision graph

Figure 2.16: Clustering a 2D example.

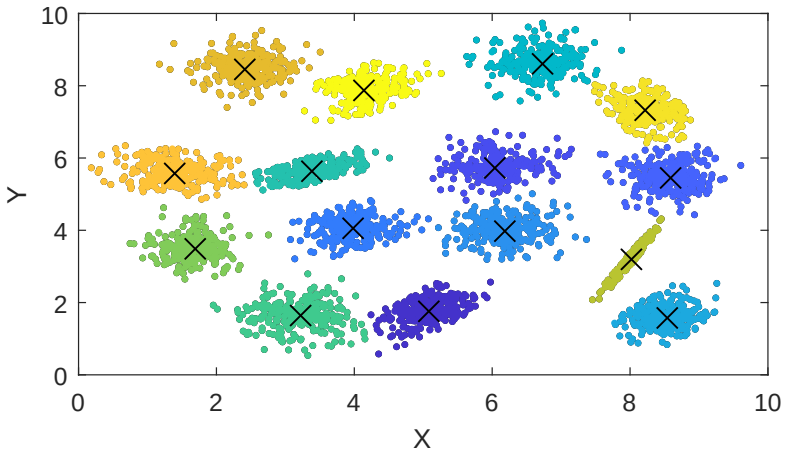
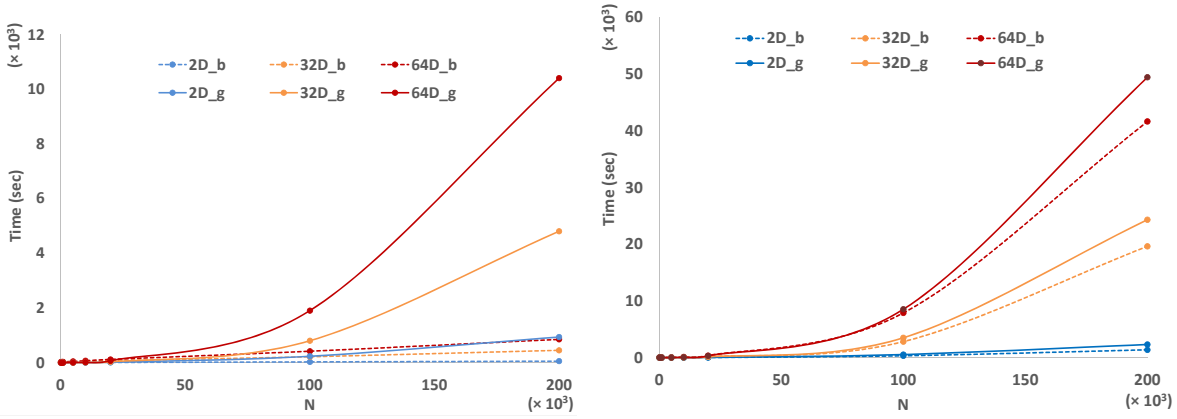


Figure 2.17: Clustered data

As stated in [89], for small datasets, computing ρ_i using the cutoff kernel described in Eq. 2.8 might be affected by large statistical errors; thus, they suggest using more accurate techniques for the density estimation part, like the Gaussian kernel described in [91]. Furthermore, this kind of distance-based density estimation at

each point i requires a scan through all the distances d_{ij}. This means that for a dataset with N samples, it will search through a total of $N \times (N-1)$ entries. This will result in huge computation time for large datasets, and the computation time will grow exponentially with N. A similar search through the distance matrix needs to be done, for computing values of δ.

In this section, as an improvement to this density-based clustering algorithm, the density estimation part of the algorithm is replaced by the BSP-based density estimator developed and described earlier in this chapter, for which we know that the execution time changes linearly with sample size N.



(a) Density estimation time

(b) Total clustering time

Figure 2.18: Execution time vs. sample size N.

A set of simulations is done to compare the performance of the modified method with the original algorithm. The data used in these simulations are generated by SynDECA [92].

Plots in Figure 2.18 make a comparison between the original method using Gaussian kernel (denoted as 'g') and the modified method using the BSP-based density estimator (denoted as 'b'), for a range of dimensions $D = 2,\ 32,\ 64$. It can be seen how, with increasing N, the density estimation time for the Gaussian kernel grows exponentially, while for the BSP-based density estimator, time grows linearly. This has resulted in a 15 to 40% reduction in total clustering time, depending on the dimensions. Clustering accuracy measures are not reported here, as there was not a meaningful difference between the two methods in that regard.

2.8.2.1 Accelerated clustering using BSP

In this subsection, I will present an algorithm for accelerated density-based clustering, using BSP density estimator. It is a variation of the density-based clustering algorithm introduced in the previous subsection, which yields a reduction in computational complexity, while retaining the clustering accuracy of the original method. The improvement in computation time is achieved by reducing the time spent on creating the distance matrix and calculating δ.

The accelerated clustering algorithm is similar to the clustering algorithm described in the previous section, with the following differences and additional steps:

1. At density estimation stage, use BSP method for partitioning the sample space.

In this algorithm, BSP does not use copula transform and instead, makes the cuts directly in the D-dimensional space, because we need to obtain the partition coordinates and their corresponding subregion centers in the original sample space.

Let us assume that the BSP partitioning process generates a total of J subregions.

2. Determine the coordinates of the central points for all subregions: C_j $(j = 1 \cdots J)$.

3. Now, each subregion center C_j counts as a hypothetical data point, representing all the data points in subregion j. These hypothetical data points are now used in computing the pairwise distances and calculating δ. Therefore, the original distance matrix (dimensions: $N \times (N-1)/2$) is replaced with a much smaller distance matrix (dimensions: $J \times (J-1)/2$).

4. The new set of data points C_j and their corresponding ρ and δ values are used for performing the clustering for subregion centers. Upon completion of this stage, all data points in each subregion j are assigned to the same cluster as their corresponding center point C_j.

To demonstrate how this algorithm works, as an example I have used $N = 5,000$ samples of the 2-dimensional data presented in Figure 2.19. Outcome of density estimation stage (with cuts made directly in the original 2D sample space) is shown

in Figure 2.20. In this figure, centers of the subregions are marked with blue asterisks. Finally, in Figure 2.21 we can see the decision graph generated by the algorithm to locate the cluster centers, and the resulting clustered data.

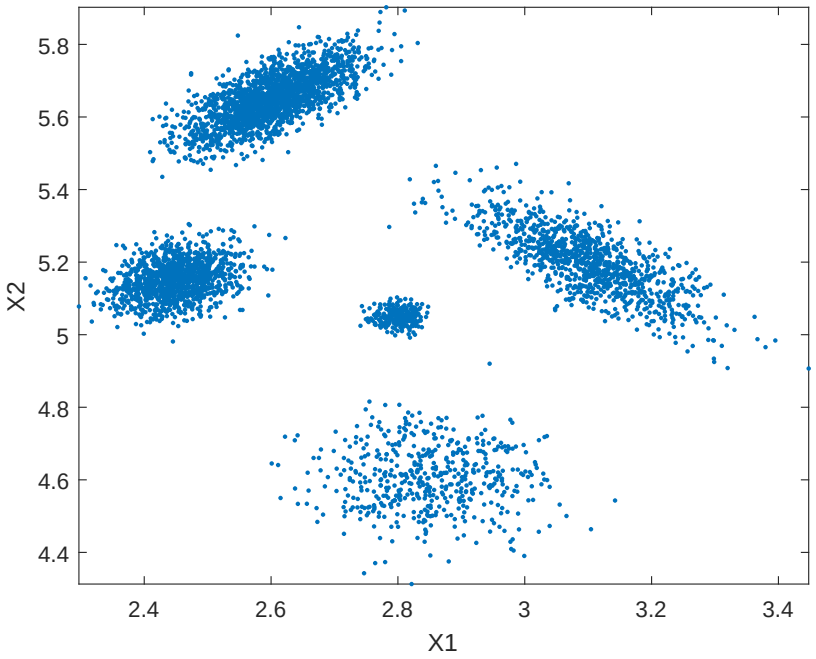


Figure 2.19: Original unlabelled data for clustering.

For this example, the accelerated method gives a clustering accuracy of 99.9%, similar to the original clustering algorithm, with 74% reduction in computation time: from 9.16 s to 2.41 s.

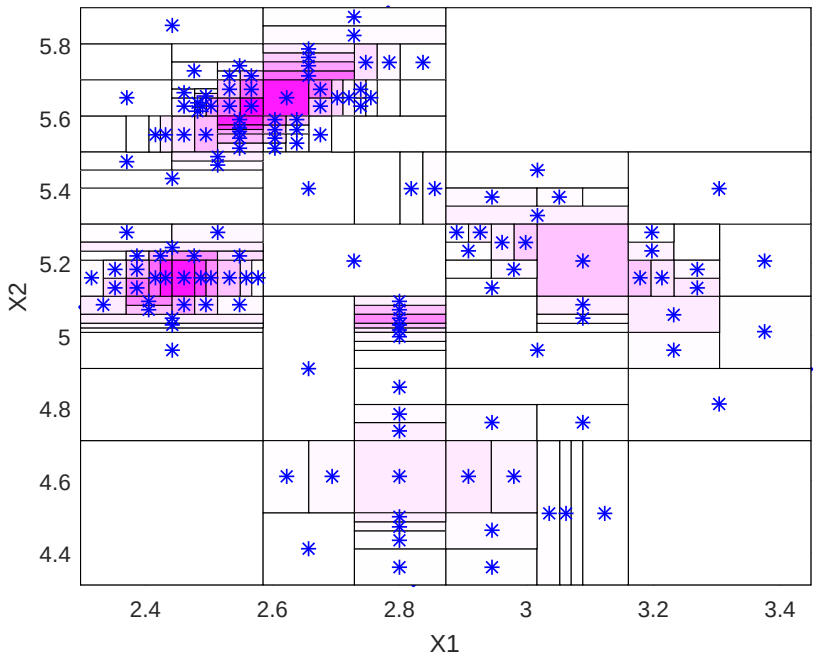
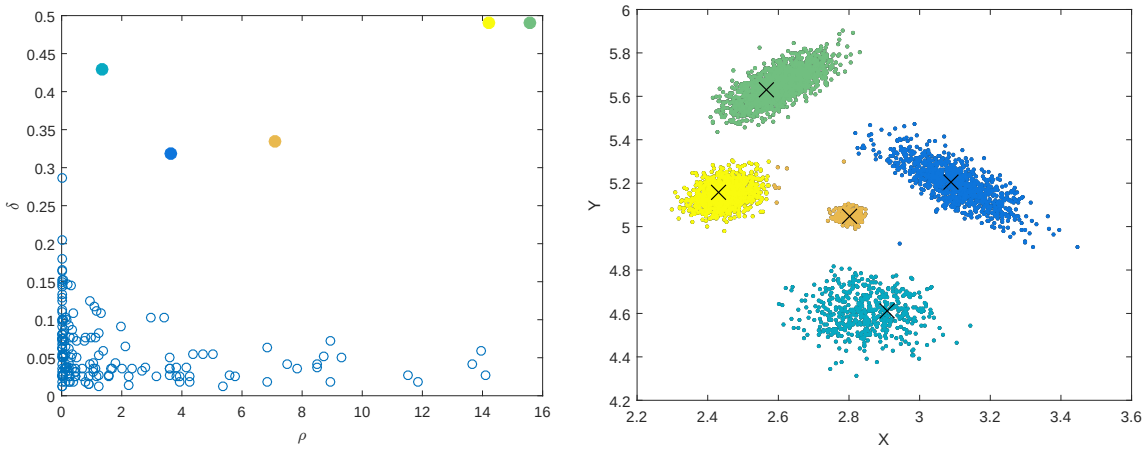


Figure 2.20: Partitioned sample space and subregion centers.



(a) Decision Graph

(b) Clustered data

Figure 2.21: Clustering a 2D example.

2.9 Conclusions

In this chapter, computational details of BSP algorithm were discussed. Also the method of copula transform to reduce the complexity of BSP was discussed. Further, a set of efficient data structures are designed for deploying BSP method for accelerated density estimation in high dimensions. All the data structures have been suitably designed for efficient implementation on parallel computing paradigms. Use of copula transform has been shown to be effective in saving significant amount of computation time, in high dimensions. It reduces the number of required cuts in the high dimensional sample space, by mapping the random variables to a copula domain, in which the data have uniform marginal distributions. Performance of the BSP algorithm, in terms of the estimation error and computation time, was investigated for a wide range of sample sizes N, and a number of sample partitions M.

Simulation results showed that computation time changes in a linear manner with N and M. Careful choice of M results in a great deal of saving in computation time. With the use of copula transform, increasing the number of sample partitions beyond $M = 1$ in estimating the marginal densities does not result in appreciable decrease in estimation error.

With separation of marginal and joint density estimation using copula transform, the idea of using KDE method or median-based cuts for computing the marginal densities was tried. Neither of these methods showed a consistent advantage over the regular

BSP method.

Further, it was demonstrated that it is possible to take the advantage of independent nature of density estimation across the marginals to fully parallelize the computation across the marginals.

Finally, sample applications of density estimation (using BSP) in density-based classification and clustering were presented. BSP-based Bayes classifier performed almost as well as other well-known classification methods. In density-based clustering, by using BSP as the density estimation core of the clustering algorithm, an existing density-based clustering algorithm was improved in terms of computational complexity.

Chapter 3

Online Density Estimation

3.1 Introduction

3.1.1 Background

Density estimation is defined as the process of constructing an estimate of the probability density function (PDF), from a set of observed data [15] [41] [93]. It is a fundamental part of statistical inference and serves as the building block of many other data mining and machine learning techniques [4] [5] [7]. In high-dimensional spaces, where other commonly used non-parametric methods like kernel density estimation (KDE) [94] fail, Bayesian sequential partitioning (BSP) algorithm [9] has

shown promising results.

In many practical applications today, there is a need to process data in stream, as they arrive, as opposed to processing the stationary data that is stored in a database. Traffic control, satellite monitoring and sensor networks are some of the examples [95]. A class of online learning techniques aim at predicting the outcome of a variable using its past outcomes [96] [97] [98]. Another group of online learning protocols do not rely on inter-dependencies between data instances and focus on estimating the density of the data, by only using the instances of data [96]. With the growing availability of huge amounts of data from various sources, the area of density estimation over data streams has become an attractive topic of research. Previous studies in this area have been focused on adapting the existing density estimation methods for use over data streams. In general, parametric methods are not suitable in high-dimensional domains, due to *curse of dimensionality* [15] [17], where the number of parameters rapidly increases with the sample size and the dimension. Therefore, this work focuses on non-parametric methods only.

3.1.2 Review of other methods

The most commonly used non-parametric estimation method, KDE, is at the core of many of the previous works on data streams. To adapt the KDE method to estimation over stream of data, the work in [99] [100] introduces the concept of *M-Kernel* which is basically a kernel over a moving window of data points. M-Kernel, which is based on utilization of Gaussian kernel, allows for a fast density estimation using a fixed amount of memory, regardless of volume of the data.

The concept of *Cluster Kernels*, (an extension to M-Kernels), was introduced in [101] [102]. Unlike M-Kernels method, Cluster Kernels are constructed based on the Epanechnikov Kernel [103], which has bounded support, and thus, lowers the evaluation cost. The Kernel-based method in [104] presents an adaptive version of Parzen window, *viz. Tiled Parzen Window* (TPW). In TPW, window size is adaptively changed proportional to the data arrival rate. The work also proposes a new strategy for discarding outdated data in the stream, using the derived probability density function. Further, in [105], the traditional wavelet density estimator (WDE) is adapted to *compressed cumulative wavelet density estimator* over data streams. A more recent wavelet-based online estimator is proposed in [106]. The recursive approach in this work is based on an iterative updating process of wavelet coefficients, using a sliding window.

All methods described above only deal with one-dimensional data streams. The work

in [106], however, proposes an extension to higher dimensions, but provides no experimental results in support of its applicability to higher dimensions.

In [107] [108], a method for mining data streams is developed, using Hoeffding decision trees. In another work by the same authors [109], Expectation Maximization algorithm is accelerated via minimizing the number of samples used in each step of the learning algorithm. Other methods [110] try to catch up with the streaming data, by storing a compressed model of the data, instead of storing all the observed samples.

In this chapter, a framework for online density estimation for high-dimensional data streams, based on BSP will be presented. First, the original BSP algorithm is adapted to density estimation over the *data-blocks*, and then this idea is applied to density estimation over data streams. A set of simulations is performed to analyze different features of the *blockized* BSP. Those features are then used to evaluate the proposed method, against a set of well-established design criteria. For a system with the mission of online processing on open-ended data streams, the general design criteria [111] are listed below. These criteria have been widely used as metrics for the evaluation of methods of processing data streams, [112] [113] [114].

1. The maximum processing time per record should not exceed a constant time, determined by the data arrival rate.

2. The maximum memory required to hold the input data, for the processing, and

to hold the processed output should not exceed a fixed amount, regardless of the volume of the data that has been received.

3. The processing system should be able to build a model using at most one scan of the data as they arrive, with no need for revisiting the old data.

4. The processing system must be able to produce usable results at any point in time, as opposed to only after consuming the whole data. This is because the incoming data stream may be never ending.

5. The processing system must have the ability to produce an estimate that is equivalent or nearly identical to the one that would be obtained by the corresponding offline data mining algorithm.

6. With a change in the statistics of the incoming data stream, the model at any time should be up-to-date, but also include all the information from the past that are not outdated.

The rest of the chapter presents a description of my approach to online density estimation, and discusses the performance and efficiency of the proposed framework, based on the reference design criteria [111] listed above. Section 3.2 provides a description of my proposed *Blockized BSP* (BBSP) algorithm. Section 3.3 focuses on performance analysis of the blockized density estimation, with respect to estimation accuracy, computation time and memory requirement, and describes the process of

deciding the optimum block size, in non-streaming applications. Section 3.4 then extends the block averaging algorithm developed in previous sections, to streaming applications, and compares its features against the aforementioned design objectives [111]; it starts with stationary streams in the first subsection, and then generalizes the framework to non-stationary streams. Finally, the concluding remarks are presented in Section 3.5.

3.2 The algorithm

3.2.1 Bayesian sequential partitioning

The method of BSP [9], as described in Chapter 2, constructs a multidimensional histogram by making sequential binary cuts in the sample space. Consider a D-dimensional dataset $\boldsymbol{X}$ with N instances. In the method of BSP, the sample space is progressively divided into subregions. After a certain number of cuts, the density in each of the divided subregions is simply the normalized ratio of the total number of points that are located in that subregion, to its volume. The algorithm follows a binary partitioning (BP) scheme, *i.e.* each cut at a given level j splits one of the existing subregions into two equal halves. To improve the quality of density estimation, M independent paths (*sample partitions*) are tried at each level [9].

This process of sequential cuts is repeated until either the best possible partition is obtained, or the number of cuts reaches the maximum value set by the user. Once the optimum partition is obtained after t cuts, probability density is estimated for each subregion $1 \leq p \leq t$ (a D-dimensional bin), as $n_p/(Nv_p)$, where n_p denotes the data count in subregion p and v_p is the volume of that subregion.

As mentioned before, at each level the BSP process described is repeated M number of times and at the end, the resulting partition with the highest *partition score* (to be described later) is maintained as the final result [9].

Complete description of the BSP algorithm and its implementation details are available in Chapter 2.

3.2.2 Blockized BSP

Consider N instances of a D-dimensional dataset. In regular BSP algorithm [9], the entire dataset is used as the sample space Ω, which is then partitioned into subregions, and eventually the data count and volume of each subregion is used to produce an estimated density in each subregion. In the blockized BSP algorithm (BBSP), in the model development stage, first the sample space is equally split into B data-blocks. Each block b can then be considered as a different approximation of the actual sample space, $\Omega^{(b)}(b = 1, .., B)$, with each containing $L = N/B$ data

instances. Each block is processed independently, using BSP, and B sets of subregion coordinates and their corresponding estimated densities are obtained. These sets in fact represent B different estimates of the underlying probability density function.

In the test stage, for each instance of the test data, $\mathbf{z}$, it is mapped onto each of the B partitions to find its location in one of the subregions. Next, the corresponding estimated density $\hat{f}_b(\mathbf{z})$ is obtained for each of the blocks. At the end, the final value for the estimated density $\hat{f}(\mathbf{z})$ is obtained by taking the average of the estimated densities $\hat{f}_b(\mathbf{z})$ over all B blocks.

$$\hat{f}(\mathbf{z}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\mathbf{z}) \tag{3.1}$$

However, it should be noted that, in general, the dimensional extents of the sample spaces for B data-blocks are different from each other; *i.e.,*

$$\Omega^{(b)} \neq \Omega^{(b')} \tag{3.2}$$

For the averaging in Eq. 3.1 to be meaningful, all the blockized density estimations should have the same dimensional extensions and be made over the same sample space. Otherwise, averaging over blockized densities $\hat{f}_b(\mathbf{z})$ to compute the overall density function would not generate valid results, because the average densities would not integrate to unity over the entire sample space.

In my use of the BSP to estimate each of the blockized densities, the process in multidimensional space is divided into two separate steps. First, the estimations for all marginal densities are performed. Using these marginal densities, their cumulative distribution functions (CDFs) are obtained. Second, marginal CDFs are used to construct a multidimensional density partition in the standard copula C [41] sample space. It's a D-dimensional sample space over $[0,1]^D$, with uniform marginal distributions. Since BSP cuts are made based on non-uniformity of distribution in subregions, the marginally uniform copula sample space reduces the number of computationally expensive cuts in D-dimensional subregions.

Using the marginal densities, and densities obtained in the copula-transformed multidimensional sample space, the overall densities for a test data $\mathbf{z}$ is computed as,

$$f(\mathbf{z}) = c(F_1(z_1), \cdots, F_D(z_D)) \prod_{d=1}^{D} f_d(z_d) \tag{3.3}$$

where f_d are the marginal densities, F_d are the corresponding marginal CDFs, and c is obtained by taking the derivative of the copula C [41]. It should be noted that non-identical sample spaces due to different dimensional extensions for B data-blocks is only an issue for the marginals. Since the CDF is by definition bounded between 0 and 1, the issue does not exist for the transformed copula domain. Therefore, in the averaging process over B blocks, we only need to align all the B sample spaces in each of the D marginals. Since all marginal densities are in one-dimensional space, an efficient procedure to enforce the dimensional alignment across all the B blocks,

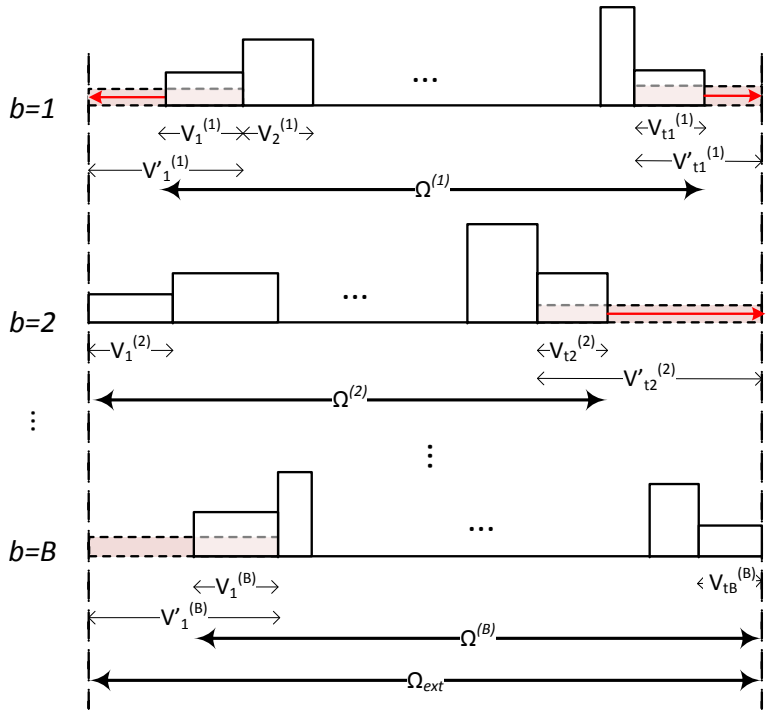


Figure 3.1: Equalizing the marginal sample spaces for all blocks $b = 1, 2, ..., B$. For each block b with the total number of t_b subregions, the first and last subregion volumes V_1^b and $V_{t_b}^b$ are properly extended to $V_1'^b$ and $V_{t_b}'^b$.

for each dimension d is to,

- Find the minimum and maximum extents of the data, among all B blocks.
- Set these extents as the common boundaries of the one-dimensional sample space for all B blocks.
- Extend the first and last subregion of each of the blockized partitions, if necessary, to be aligned with the new boundaries.

- Recalculate the marginal densities in the modified subregions, using the adjusted volumes. Note that the number of data points in these subregions remain unchanged.

- Use the adjusted marginal densities to compute the corresponding CDFs.

The above process is illustrated in Figure 3.1, where the sample spaces for all B blocks, $\Omega^{(1)}$, $\Omega^{(2)}$, ..., $\Omega^{(B)}$ are properly extended to a common sample space, Ω_{ext}. After these steps, the adjusted CDFs can be used to perform the second step of the density estimation, in the D-dimensional space. Finally, the marginal densities and copula-transformed densities are used to compute the blockized density for each of the blocks. These adjusted densities from all of the blocks can then be plugged into Eq. 3.1 to generate a valid estimation of the probability density function.

3.3 Performance Analysis of BBSP

In this section, the performance of the blockized BSP in high-dimensional space is analyzed, using some simulation results. The basis for performance measures are,

- Estimation error

- Computation time

- Memory requirement

Results obtained from these analyses will be used later to evaluate the proposed framework against the general design criteria listed in Section 3.1. Simulations are performed on a platform with an Intel i7-3820 CPU (3.60 GHz, with 32 GB RAM). To cover a range of datasets, the BBSP model is developed for two cases. The first case corresponds to a synthetic dataset with a simple structure. The second case corresponds to a real dataset with a complex structure.

3.3.1 BBSP for synthetic dataset with simple structure

First, the simulation results are presented for a sample case of a 64-dimensional synthetic dataset with $N = 400,000$ instances, randomly generated from a Gaussian distribution. The dataset used in these simulations has a relatively simple structure with a trimodal normal distribution for the first two dimensions. The third dimension is a unimodal normal distribution, and dimensions four and higher have a bimodal normal distribution. Various block sizes, from $L = 10$k to $L = 80$k have been used in the experimentation. The structural simplicity of this dataset, results in a relatively small number of cuts in the copula-transformed multidimensional sample space, and therefore, low computational complexity.

The evaluation criteria Kullback-Leibler divergence (KLD) [41] is used as a measure of the estimation accuracy with respect to the true density. Figures 3.2 to 3.8 show

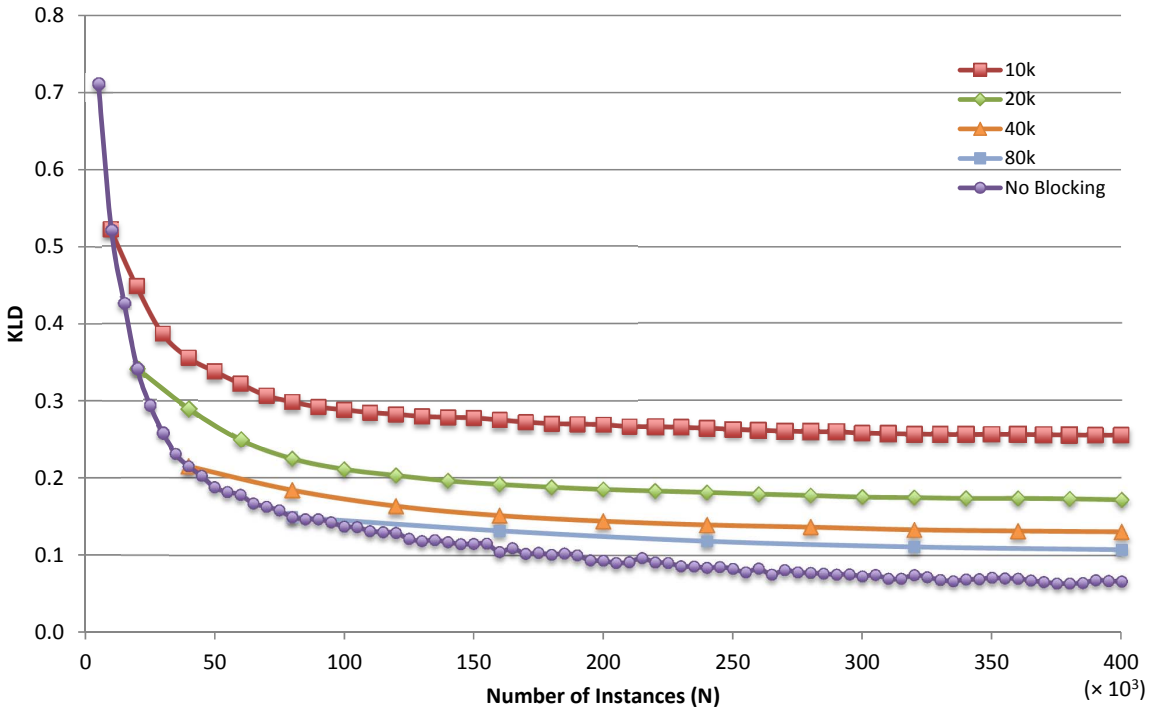


Figure 3.2: KL divergence (KLD) for various block sizes

the simulation results. The case with no blocking shows the results obtained from performing the regular BSP algorithm with no blocking, *i.e,* using the entire available dataset as a single data-block, with a block size of $L = N$. For these results a value of $M = 200$ is chosen. A number of useful observations can be made from the results presented in these figures.

- As shown in Figure 3.2, the cases with smaller block sizes start at larger values of KLD (higher estimation error).

- From Figure 3.2, all KLD plots reach a saturation point with certain number of blocks, after which, adding more blocks will not improve the estimation accuracy.

- Again from Figure 3.2, through the entire range of N i.e., $N \leq (N_{max} = 400,000)$, the KLD plot for the case with a larger block size lies underneath the case with a smaller block size, with the no blocking ($L = N$) case having the best KLD rating across the whole range of 10k $\leq N \leq$ 400k.

- With the same total number of instances N, the overall time for obtaining the density estimation over B blocks of data, each having a size of $L = \frac{N}{B}$ is the sum of processing times for all B blocks, plus some overhead time for averaging the densities obtained from the blocks. The latter is a small fraction of the overall time and can be ignored. Therefore,

$$t_L(N) = \sum_{b=1}^{B} t_b(L) + t_{overhead} \approx \sum_{b=1}^{B} t_b(L) \tag{3.4}$$

where $t_L(N)$ represents the total time for processing $N = BL$ instances, using B blocks of size L each, and $t_b(L)$ is the time for processing the b^{th} block.

- As Figure 3.3 indicates, for any block size L, the block computation time is almost constant for all B blocks. Further, Figure 3.4 presents the relationship between the computation time and the block size. Therefore, assuming equal processing time $t_{avg}(L)$, for all B blocks, the overall processing time can be expressed as,

$$t_L(N) = B \times t_{avg}(L) = \frac{N}{L} \times t_{avg}(L) \tag{3.5}$$

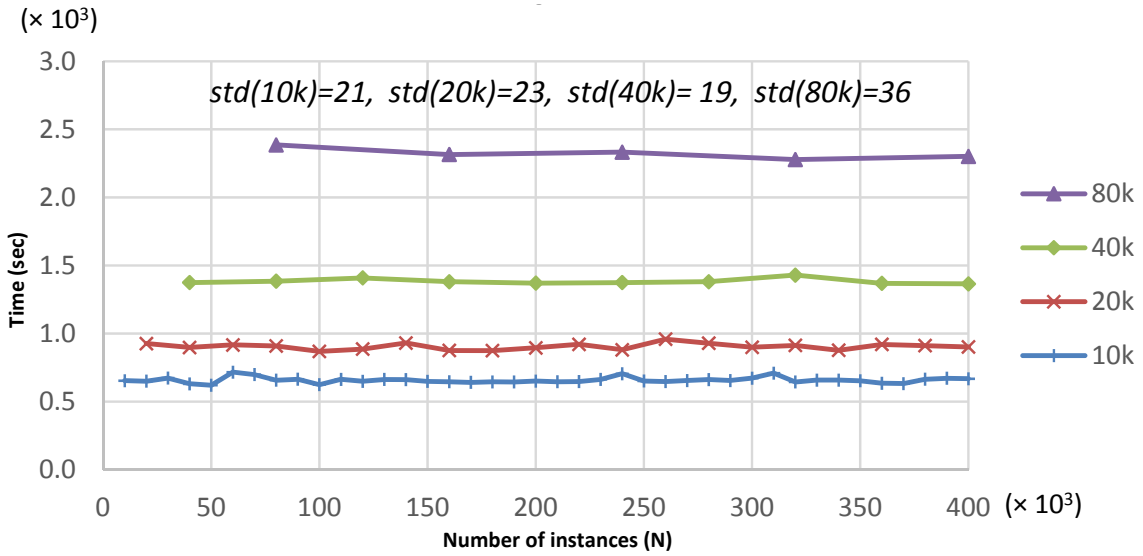


Figure 3.3: Computation times for individual blocks, for various block sizes. The standard deviations are reported on top of the plots.

The timing plot in Figure 3.4 is slightly sub-linear and therefore, for two cases with block sizes of L_1 and L_2, where $L_2 > L_1$, we can write,

$$t_{L_1}(N) = \frac{N}{L_1} \times t_{avg}(L_1)$$

$$t_{L_2}(N) = \frac{N}{L_2} \times t_{avg}(L_2)$$

$$\frac{t_{L_2}(N)}{t_{L_1}(N)} = \frac{\frac{t_{avg}(L_2)}{L_2}}{\frac{t_{avg}(L_1)}{L_1}} < 1$$

$$t_{L_2}(N) < t_{L_1}(N)$$

- Similarly, from Figure 3.5 and the plot for memory requirement[1] in Figure 3.4,

[1]In all plots the memory is reported in units of storage for double precision floating point (8 bytes) numbers.

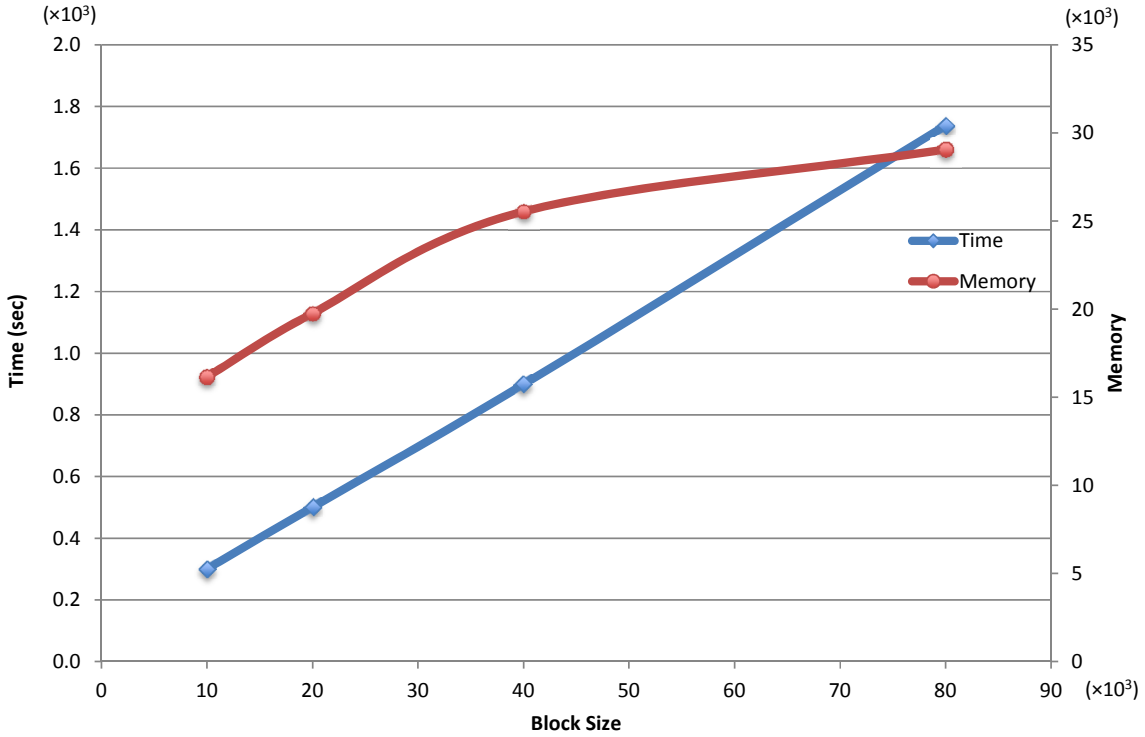


Figure 3.4: Relation between computation time and the required memory, and the block size for a high dimensional dataset with simple structure

for any block size L, the amount of memory required for storing the subregion coordinates and density values for each of $B = \frac{N}{L}$ blocks remains almost constant. Using a similar argument, for two cases with block sizes of L_1 and L_2, where $L_2 > L_1$, we can write, $m_{L_2}(N) < m_{L_1}(N)$, where $m_L(N)$ represents the total memory requirement for storing the density estimation partitions for $N = BL$ instances, using B blocks of size L each.

In summary, for a high dimensional dataset with a simple underlying structure, if the entire dataset of a certain size N is available, the case with no blocking is the most efficient choice, with respect to estimation accuracy, as well as computation time and

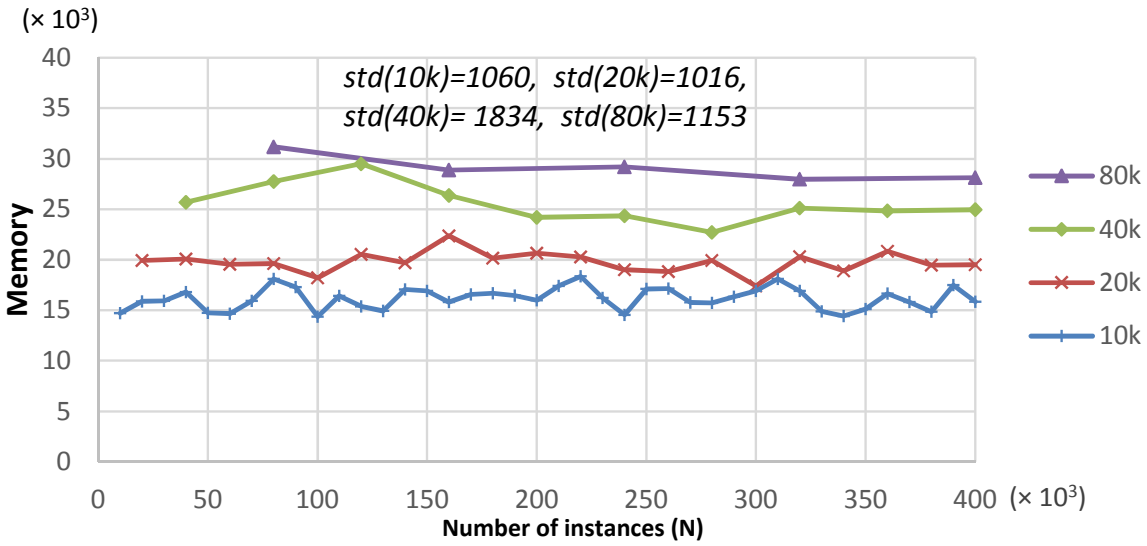


Figure 3.5: Memory requirements for storing output density information for individual blocks, for various block sizes. The standard deviations are reported on top of the plots.

memory requirement.

From the foregoing discussion, it is more meaningful to compare the performance of the algorithm for different block sizes by measuring the computation times required to reach a given target KLD. Because, as observed earlier, after a certain point, adding more data instances results in a negligible improvement to the estimation accuracy. Figures 3.6 and 3.7 show the computation times and memory usages to reach target KLD values of 0.2, 0.25 and 0.3, for different block sizes. Both plots show very similar trends; The larger block sizes reach the target KLD faster and have a smaller memory footprint. This trend continues up to the block size for which processing only one block of data will achieve the target KLD. Beyond that point, increasing the block size only results in larger computation time and memory usage.

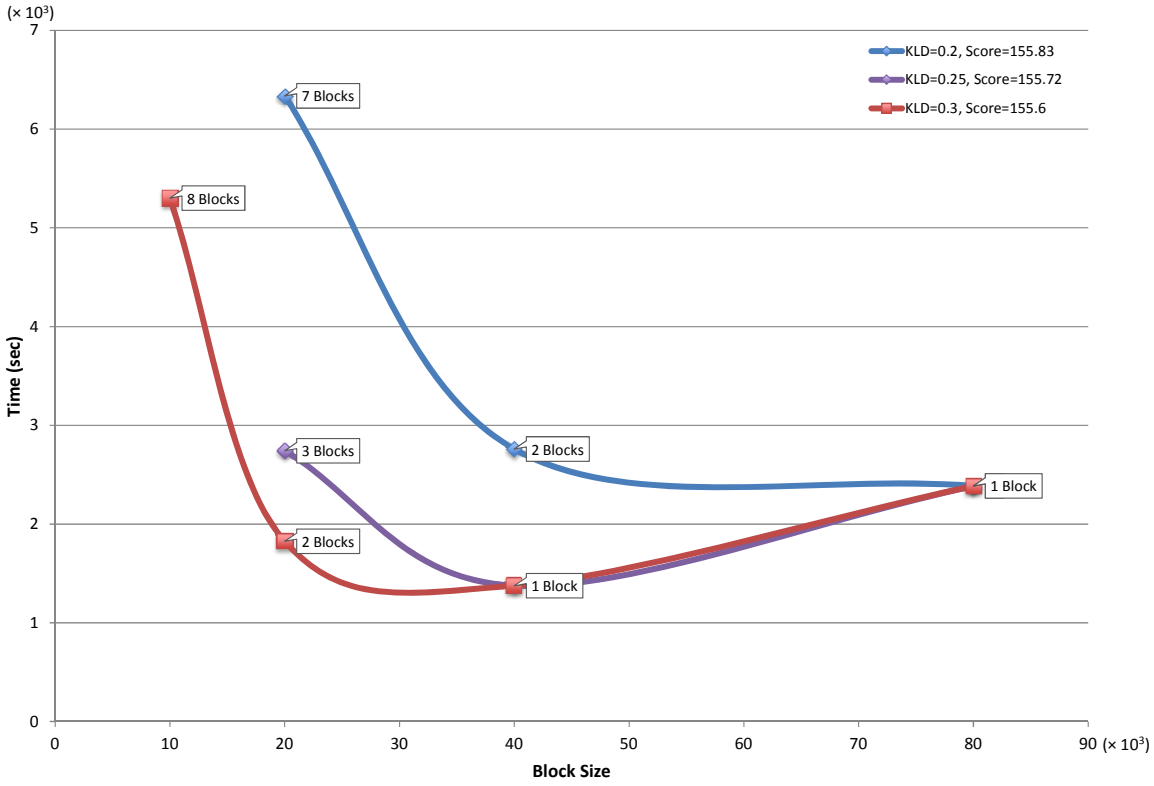


Figure 3.6: Time to reach different target KLD values. The corresponding partition scores are reported in the legend.

The plots in Figure 3.7 only present the memory footprint for storing the partition information from the individual blocks. If the memory for storing one block of input data is also included, where it actually becomes the dominant part of the overall memory usage, the plot will take the form shown in Figure 3.8. These plots clearly demonstrate the trade-off between computation time and estimation accuracy on one side, and the memory usage on the other. Use of larger block sizes leads to better estimation accuracy and lower computation time. On the other hand, obviously, the amount of memory needed to temporarily store one block of input data increases with the block size.

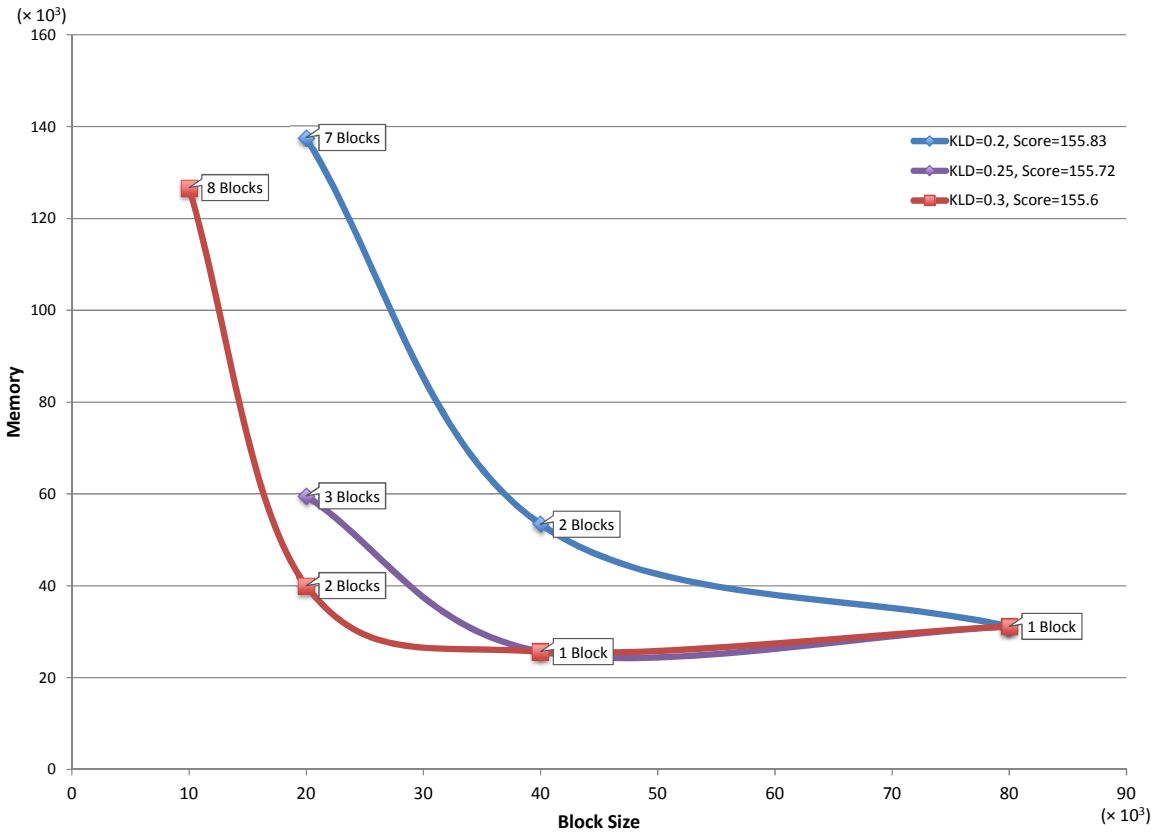


Figure 3.7: Memory requirement for storing the processed output density information for different target KLD values. The corresponding partition scores are reported in the legend.

3.3.2 BBSP for real dataset with complex structure

Next, a BBSP model is developed for a real dataset with a complex structure, where the marginal variables have high correlation. The structural complexity inherent in this dataset results in a very large number of cuts in the copula-transformed multidimensional sample space, and therefore, significantly higher computational complexity, compared to the previous case. To reduce the complexity and simulation time, I had

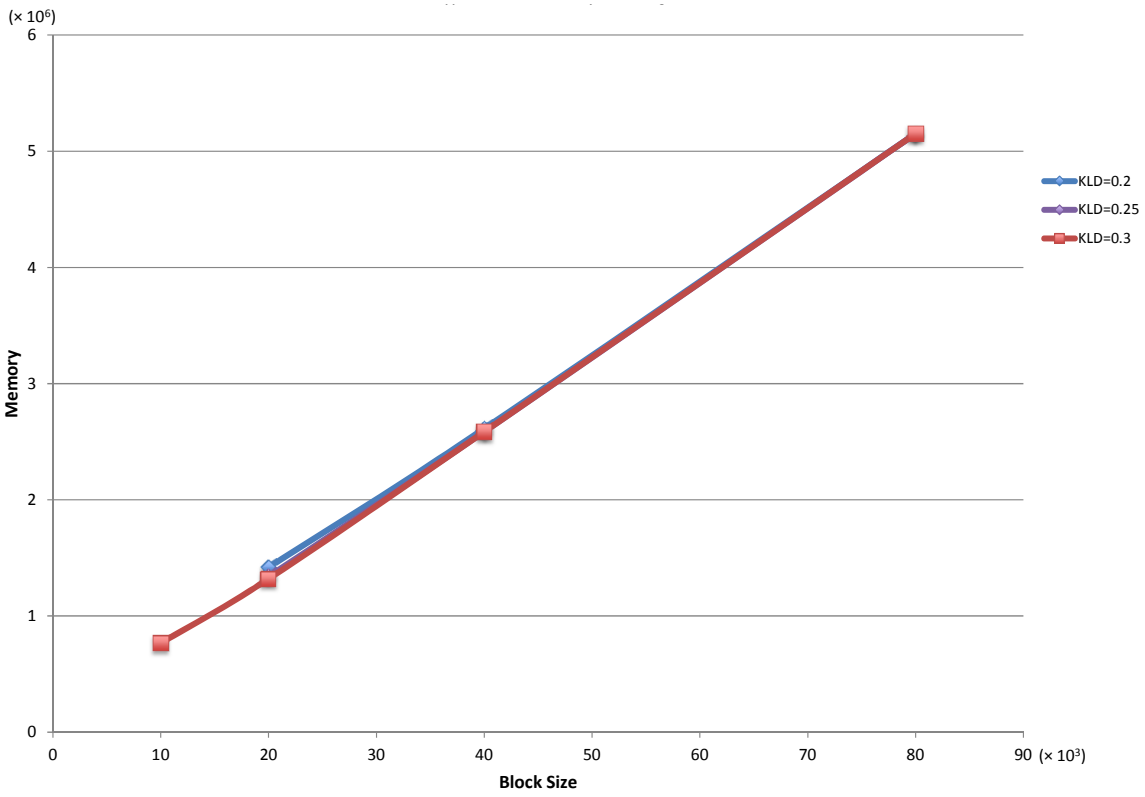


Figure 3.8: Overall memory requirement (input data-block and output density information) for different target KLD values.

to set $M = 3$ at the expense of reduced estimation accuracy.

As discussed, a measure of KLD between the true and estimated PDF provides us with the means to choose the best block size to achieve a target KLD. Plots in Figures 3.6 and 3.7 (and in fact Figure 3.2) are from synthetic data with simple structure where the true densities are known, and thus, the KLD between the true and estimated PDF can be calculated. However, for the practical cases with real datasets, there is no access to the true densities to compute a measure of KLD. To resolve this issue, the original BSP algorithm in [9], defines a *partition score* for each of the M sample

partitions.

For a partition p, consisting of j subregions, partition score is defined as follows [9];

$$score(p) = -\beta j + \log(\frac{D(n_1 + \alpha, ..., n_j + \alpha)}{D(\alpha, ..., \alpha)}) - \sum_{k=1}^{j} n_k \log(|V_k|) \tag{3.6}$$

where α and β are constants, n_k is the data count in subregion k, and V_k represents its volume. $D(.)$ refers to the Dirichlet distribution [115], which is a multivariate generalization of the Beta distribution [115].

The defined partition score divided by the number of data instances N, hereafter denoted as PSN (Partition Score, Normalized), is shown [9] to linearly increase with the reduction in the computed KLD. Therefore, for real datasets we can replace KLD with PSN, for estimation accuracy evaluations.

To develop an appropriate BBSP model for the real datasets, the 90-dimensional "Year Prediction MSD Dataset" from the University of California Irvine, Machine Learning Repository [77] is used, as an example. It contains the release year and timbre audio features of over 500,000 songs, produced between 1922 and 2011. I have used 400,000 instances of this dataset for my BBSP model development.

Figure 3.9 presents the plots for the KLD and the PSN for a range of block sizes. Since the true densities are not known, the KLD values are computed as measure of

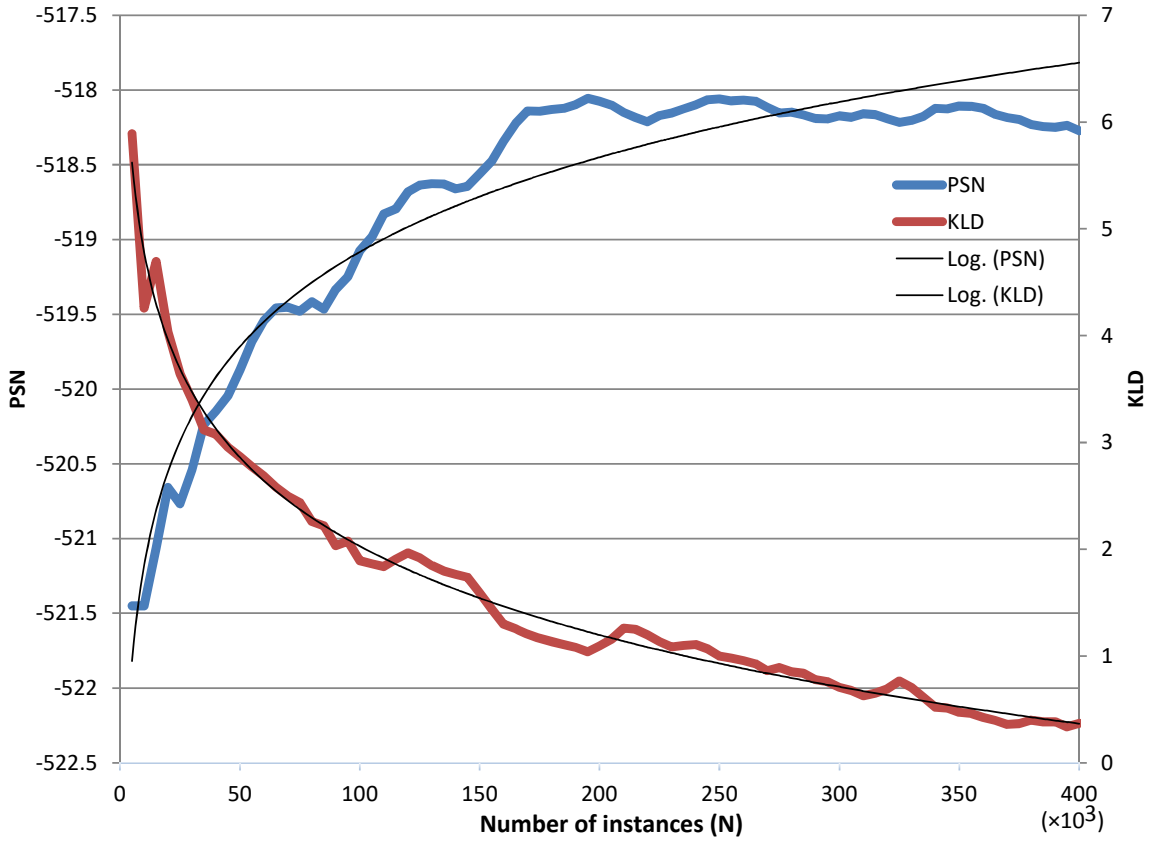


Figure 3.9: KLD and normalized partition score over N (PSN) versus sample space size

divergence between the estimated densities corresponding to each block size and the estimated densities obtained from the entire 400,000 data instances. As expected, an increase in the sample size, results in an increase in PSN and a corresponding decrease in the KLD. As seen, both plots reach saturation around the same block size.

The linear relationship between PSN and the KLD is depicted in Figure 3.10. Therefore, a choice of a particular target KLD can be replaced with an equivalent target

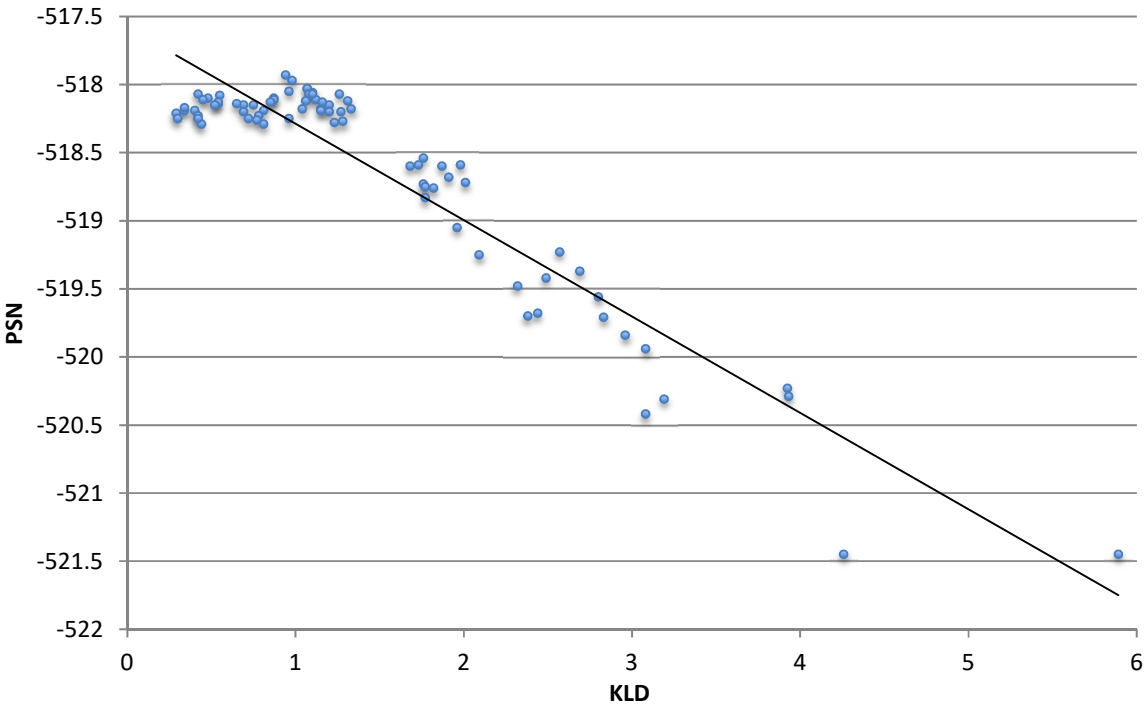


Figure 3.10: Linear relationship between partition score over N (PSN) and KLD

PSN as a measure of accuracy in the density estimation. However, the range of numerical values for the KLD and the PSN varies with the size and the underlying distribution of dataset. Thus, to have a better measure of a target PSN that can be used more universally across a wide range of datasets a normalized PSN, hereafter denoted as dPSN (differential PSN), is used as the ratio of the incremental change in the PSN to the actual value of the PSN at a given block size. This measure is shown in Figure 3.11 for the"Year Prediction MSD Dataset". As can be seen, dPSN rapidly reduces with the block size.

So, to develop a BBSP model for the real datasets similar to that in Figure 3.6, we

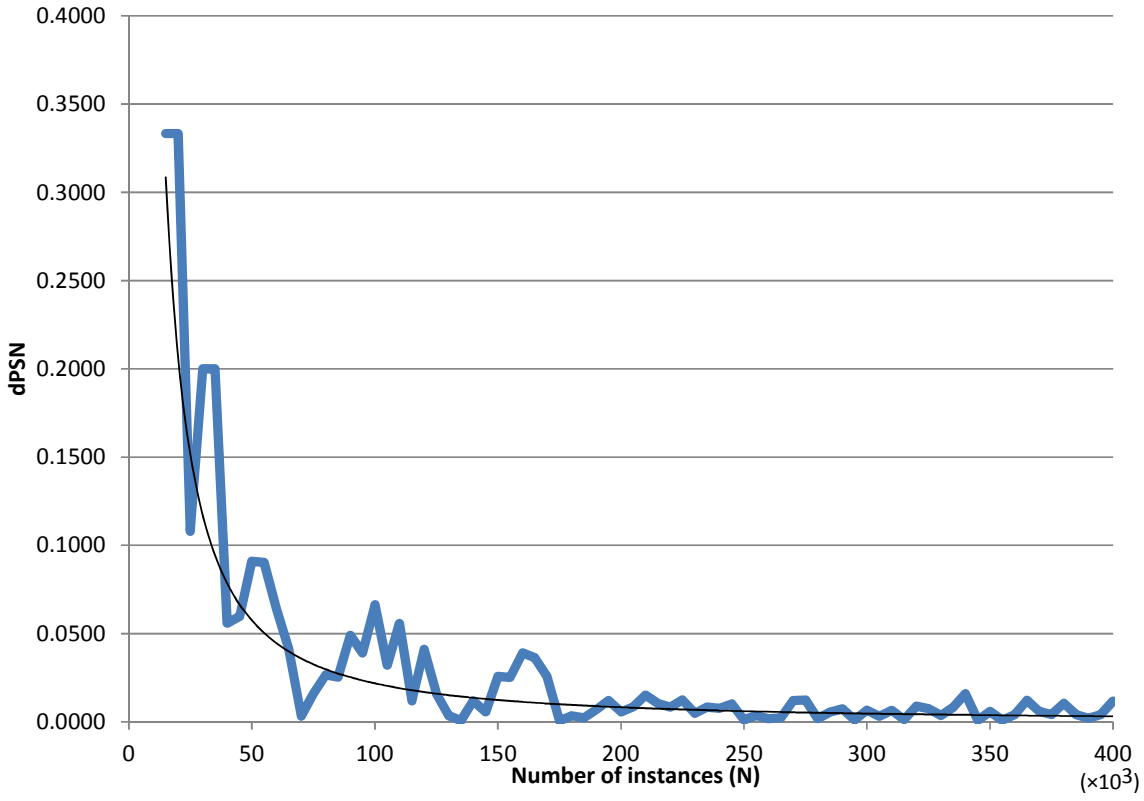


Figure 3.11: Differential normalized partition scores over N (dPSN) versus the data-block size

follow the steps below,

- For a given dataset, choose a target dPSN, and determine its corresponding block size from Figure 3.11. This block size corresponds to the smallest single block that achieves the target dPSN. This point corresponds to the point marked as "1 Block" on the plot in Figure 3.6 which has the desired target partition score (or KLD).

- For the target dPSN, choose a range of block sizes, and for each, determine the number of blocks that are required to maintain the same level of estimation

accuracy that was obtained for the single block size obtained in the previous step. For a given dPSN, this corresponds to a trace-back along one of the plots in Figure 3.6 from the "1 Block" point. To achieve this trace-back, for each block size we include required number of blocks in the density estimation averaging in Eq. 3.1 to achieve the same estimation accuracy as that of the single block case. For each block size, KLD between the estimated density averaged over the number of blocks, and the estimated density obtained from the "1 Block" case can be used as the measure of accuracy deviation. Figure 3.12 depicts the trace out plot for dPSN = 0.025.

The shape of plot in Figure 3.12 for a complex data structure is significantly different from that of a simple data structure of Figure 3.6. Here, the choice of the smallest block size is computationally most efficient. This can be explained through the fact that the complexity, in terms of the number of cuts in the copula-transformed multidimensional sample space (and therefore, the overall computation time), grows rapidly with the block size. This relation, as shown in Figure 3.14, presents a striking contrast to the sub-linear relation in Figure 3.4. As a counterpart to the case of synthetics data with simple structure, it can be stated that for two block sizes of L_1 and L_2, where $L_2 > L_1$, we can write $t_{L_2}(N) > t_{L_1}(N)$. The figure however, shows a sub-linear relation between the memory usage and the block size, and therefore, $m_{L_2}(N) < m_{L_1}(N)$, when $L_2 > L_1$.

Figure 3.13 provides a more detailed analysis of the memory requirement for a dataset with a complex structure. First, unlike the case of a dataset with simple structure in Figure 3.8, the memory requirement for storing the partition information is comparable to the memory required for storing a block of 90-dimensional input data. However, memory required for storing the partition information follows the same pattern as was observed in Figure 3.7. This memory requirement reaches to its minimum value at the block size of 80k before it starts increasing again (not shown in Figure 3.13). The overall memory requirement like the case in Figure 3.8, increases with the block size. However, the memory required is twice larger.

It is concluded that for a high dimensional dataset with a complex structure, there is a trade-off between computation time and memory usage on one side, and the estimation accuracy on the other. Use of larger block sizes leads to better estimation accuracy. On the other hand, the computation time and the amount of memory needed for the partition information and to temporarily store one block of input data increases with the block size.

For each specific application requiring online density estimation, the decision on the optimum block size can be made based on the requirements of that application, in terms of estimation accuracy, as well as available computing resources and storage space. Depending on the priorities, the most critical parameters are decided first, and then the remaining parts of the design need to be determined accordingly.

The decision process for determining the size and number of blocks can be summarized as follows:

1. Use the plot of $dPSN$ versus N (Figure 3.11), to determine the number of samples corresponding to a saturation point in the plot. This is the point beyond which, increasing number of samples would not result in significant change in $dPSN$ (and accordingly, estimation error).

2. Based on the system's memory constraints, use the plot of memory versus block size (Figure 3.13), to put an upper limit on the block size, such that the total amount of storage required to store the data and other necessary data structures does not exceed the available memory space.

3. Constraining block size based on the available memory, next the plots of processing time versus the block size (Figure 3.12) are used to further narrow down the choices in block size, (and therefore, the number of required blocks) that can be processed within the limit of the latency.

4. Finally, among the candidate block sizes that satisfy both memory and time constraints, pick the block size (and number of blocks) that results in the smallest difference in (KLD), from the result obtained by using the maximum possible block size.

For our real dataset example, following the steps above, using Figures 3.11 and 3.12,

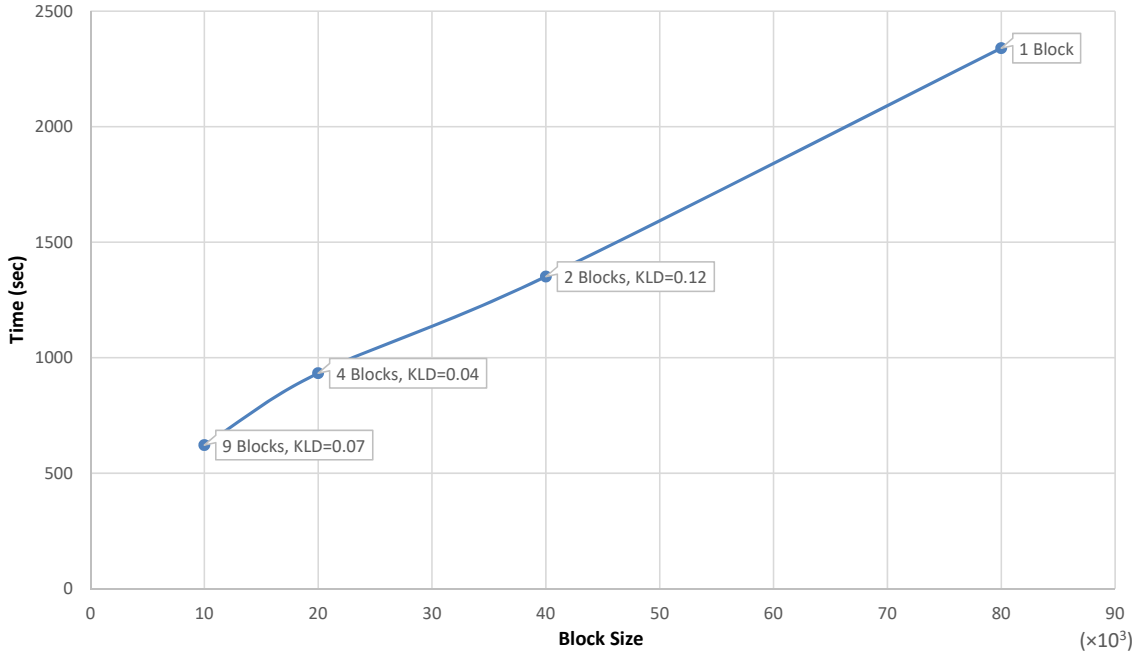


Figure 3.12: Time to reach target dPSN = 0.025, for various block sizes. The number of blocks to achieve the required dPSN and the achieved KLD values with respect to a block size of 80k are marked on the plot.

for a certain application with a given computing power, storage limit, and time constraint, leaves two possible choices. From the choice of four blocks of size 20k, or nine blocks of size 10k, the former choice of $L = 20k$, has the least KLD difference (≈ 0.04) with respect to the block size of $80k$.

3.4 Application to streaming data

In previous subsections, developing an offline BBSP model for synthetic and real datasets was discussed. Now, situations are examined in which the entire dataset is

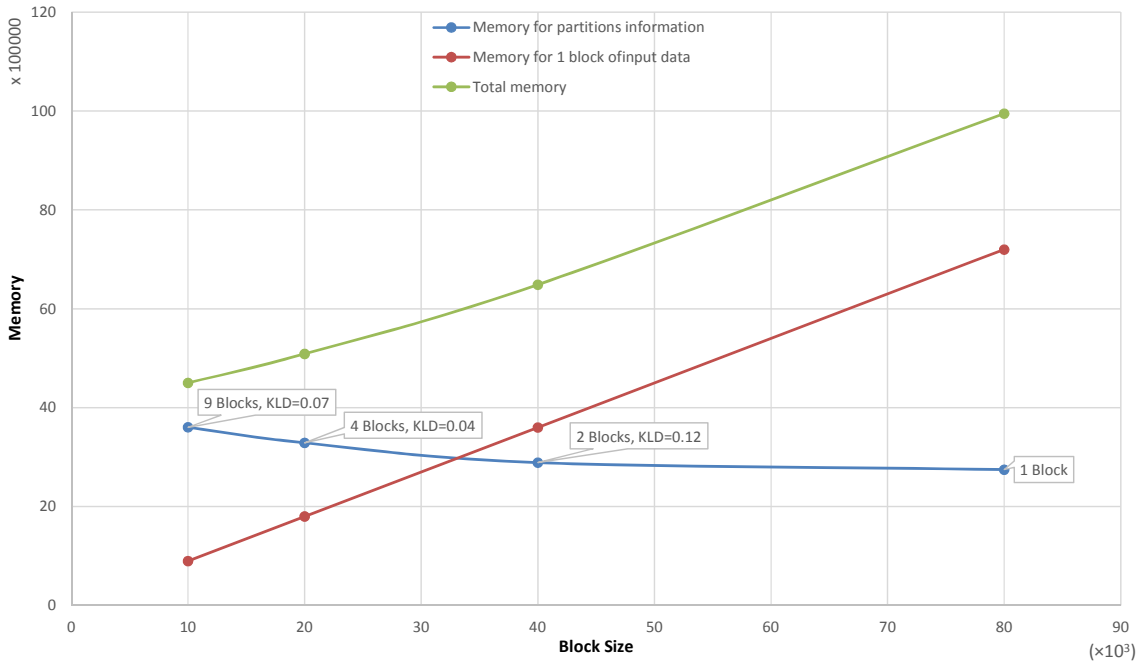


Figure 3.13: Memory requirements for dPSN=0.025, for a complex high dimensional dataset.

not available, and density estimation can only be performed on the data streams. It will be shown how BBSP model developed in the previous section can be used to design a density estimation framework over data-block streams.

First, we will discuss online density estimation over stationary streams. Then we will extend the framework to the more general case of non-stationary streams, in which the underlying density function is changing over time.

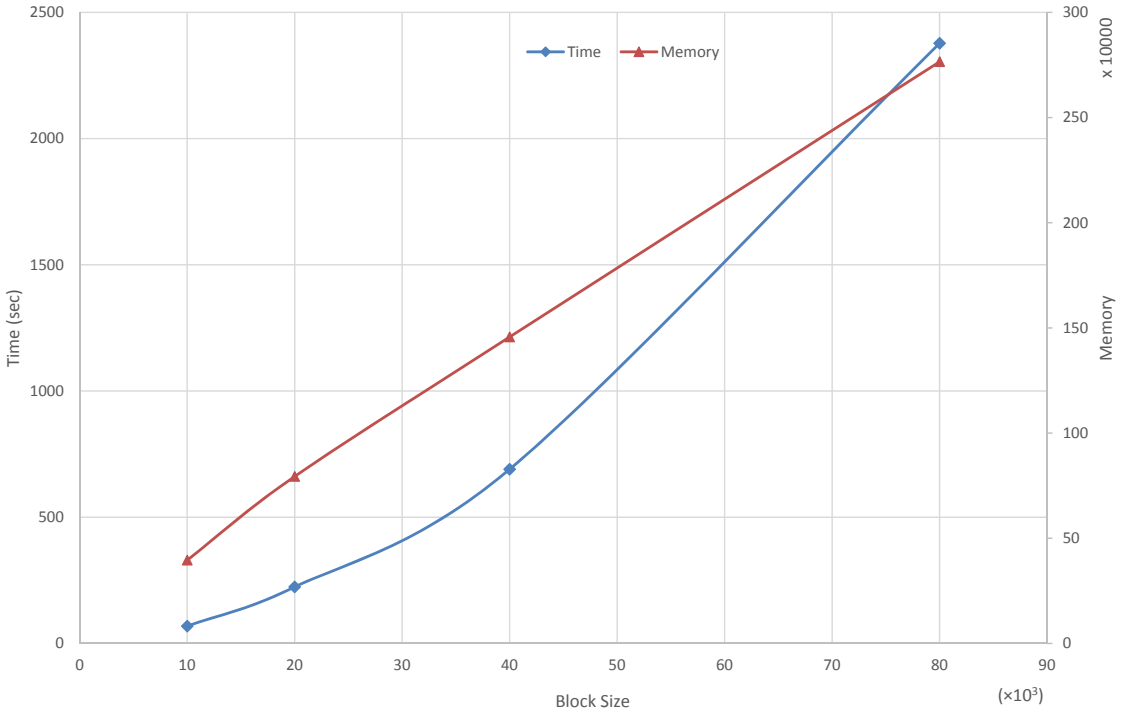


Figure 3.14: Relationship between the computation time and the required memory, and the block size for a complex high dimensional dataset.

3.4.1 Density estimation over stationary data streams

Consider an open-ended data stream with a constant arrival rate of R_a instances per second. The received data are processed at a rate of R_p instances per second. It is assumed that from the performance observations in the previous section, we can evaluate the suitability of the proposed framework to density estimation over the data streams against the design criteria listed in Section 3.1.

1. For a given fixed block size, the BBSP processing time is almost constant for

all the data streaming blocks, as shown in Figure 3.4.

2. For a given block size L, the overall memory required for storing the current input data-block being processed, the one being collected, and the output partition information is almost constant, and does not change over across the data-blocks. This overall memory requirement is approximately twice the values shown in Figure 3.8 for datasets with simple structures. For datasets with more complex structures, the corresponding required overall memory ranges 1.2 to 1.7 times the total values shown in Figure 3.13. For each processed data-block, only the coordinates of the subregions and their corresponding densities need to be stored. Also, with the number of blocks for achieving the target accuracy being known, the density information for the oldest block can be discarded, after processing of each new block.

3. Once processing of one data-block is completed, all processed partition related information is stored for the future use. At any point, the algorithm uses these information from the most recent data-blocks to compute the average density for any test data. The number of the necessary data-blocks are determined by the block size and the desired accuracy. The density estimation records for any older data-block is removed from the memory.

4. The first density estimation can be obtained after an initial wait time required for the collection and processing of the first data-block as $t_0 = \frac{L}{R_a} + \frac{L}{R_p}$.

5. As the results presented in the previous section showed, for all block sizes, the estimation accuracy improves (up to a saturation point) over time, with processing of more data-blocks. With the proper choice of block size, the estimation accuracy can be made arbitrarily close to the results from the regular (offline) algorithm with no blocking.

Thus, the proposed BBSP approach to density estimation fully satisfies the first five design criteria. Next section will evaluate the performance of the BBSP framework against the last criteria, i.e., the non-stationary aspect of the streams.

3.4.2 Density estimation over non-stationary data streams

In general, the underlying probability density for the data stream changes over time. However, in this analysis it is assumed that adequate initial data instances are available to develop a BBSP model to obtain an optimum block size as was described in Section 3.3.2. It is also assumed that for a given dataset, the temporal changes in the density distribution do not significantly alter optimum block size depicted in Figure 3.6 or Figure 3.12.

Therefore, in dealing with a non-stationary data stream, we need to be able to properly update the estimated PDF. From the KLD plots in Figure 3.2, it can be seen that the estimation accuracy does not improve after the processing of a certain number of

blocks. Thus, to reduce the memory footprint, we only need to maintain a window of the most recent blocks that must be included in the averaging of the estimated densities.

3.4.2.1 Analysis

To address the last requirement for processing the data streams listed in Section 3.1, this section provides an analysis of the situation where the statistics of the incoming data stream changes over time. Recall the open-ended data stream defined before, with a constant arrival rate of R_a instances per second, for which the received data are being processed at a rate of R_p instances per second. Now let us assume that the underlying density changes once every T_c seconds, or equivalently at a rate of $R_c = \frac{1}{T_c}$ changes per second. Two measures of performance are defined as follows.

- **Response Time** (τ_{resp}): The time it takes for the density estimator to respond to a change in the statistics of the incoming stream of data. In other words, the time gap between the point where there is a changes in the statistics and the point at which the first update on the estimated density is obtained.

- **Settling Time** (τ^{α}_{sett}): The time it takes for the density estimator to collect and process enough number of blocks, in order to reach the target accuracy (KLD $\leq \alpha$), after a change in the statistics.

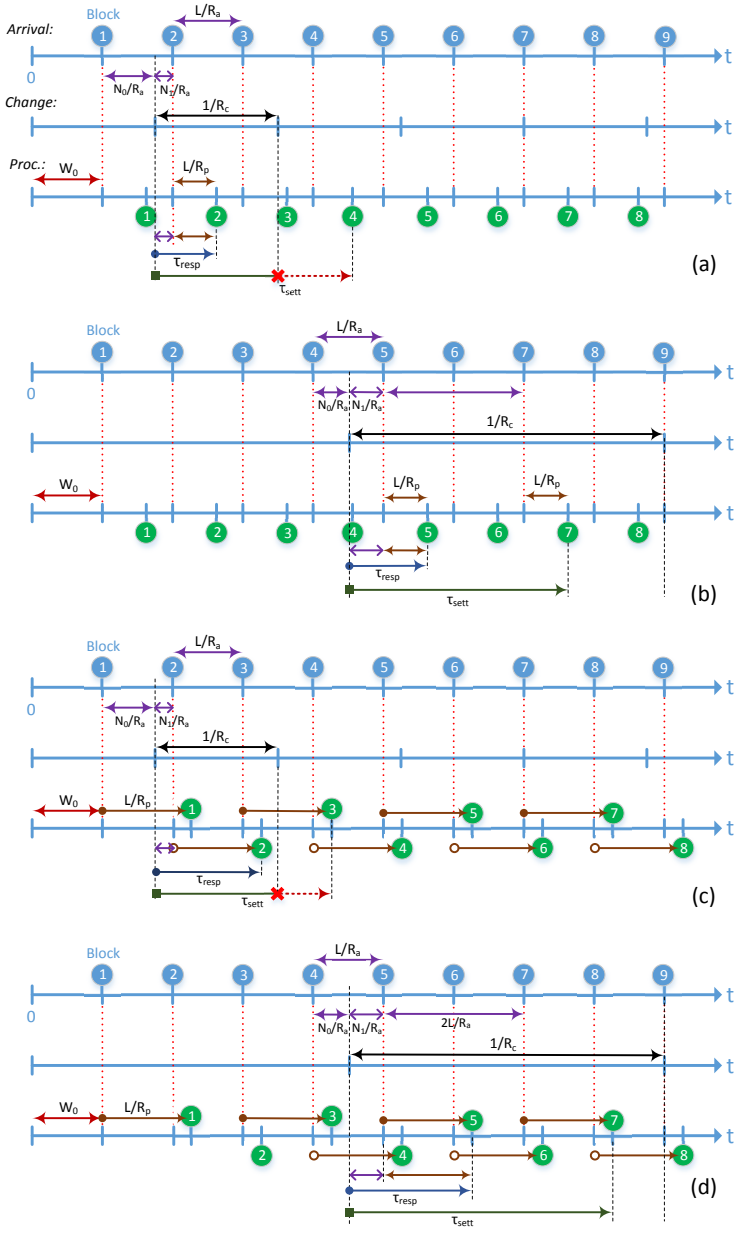


Figure 3.15: Relation between arrival rate, processing rate and change rate, with response time τ_{resp} and settling time τ_{sett}, (a) $R_p \geq R_a$ and $\frac{1}{R_c} < \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$ (b) $R_p \geq R_a$ and $\frac{1}{R_c} \geq \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$ (c) $R_p < R_a$ and $\frac{1}{R_c} < \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$ (d) $R_p < R_a$ and $\frac{1}{R_c} \geq \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$

For a set of rates, (R_p, R_a and R_c) several different scenarios can be envisaged. Figure 3.15 presents these scenarios for the example case of B_L^α=2, (*i.e.*, the case where two data-blocks of size L need to be processed to achieve the target KLD of α).

- **Scenario (1) :** $R_p \geq R_a$: The processing rate is higher than the arrival rate. Then depending on the rate of change in the statistics of the incoming data, two main cases can be considered:

  - **Scenario (1-1):** $\frac{1}{R_c} < \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$: This scenario is depicted in Figure 3.15 (a). The response time τ_{resp} is the sum of the time needed to fully receive the currently arriving data-block to its end, plus the block processing time,

$$\tau_{resp} = \frac{L - N_0}{R_a} + \frac{L}{R_p} \tag{3.7}$$

    where N_0 is the number of data instances from the current block of size L that are already received at the point of change in the statistics.

    In this scenario the value of τ_{sett}^α can not be determined, as the density estimation cannot settle to a steady state. This is because the estimator is not able to collect and process the required B_L^α number of blocks, before the next change in underlying statistics in the streaming data.

  - **Scenario (1-2):** $\frac{1}{R_c} \geq \frac{B_L^\alpha}{R_a} + \frac{L}{R_p}$: The response time, τ_{resp} in this scenario is again calculated using Eq. 3.7.

Since the changes in the statistics are occurring at a sufficiently low rate, in this scenario target KLD can be reached in between the changes. The settling time therefore can be expressed as,

$$\tau_{sett}^{\alpha} = \frac{(B_L^{\alpha} + 1) \times L - N_0}{R_a} + \frac{L}{R_p} \tag{3.8}$$

The first term is the time needed to collect the data to the end of the current block when the statistics changed, plus B_L^{α} new blocks of data with updated statistics. Figure 3.15 (b) illustrates this scenario.

- **Scenario (2):** $R_p \leq R_a$: This is an infeasible case as the data cannot arrive at the a higher rate that it can be processed. The processing rate can be increased by employing sufficient number of parallel processors, P, such that $(R_p \times P) \geq R_a$.

  - **Scenario (2-1):** $\frac{1}{R_c} < \frac{B_L^{\alpha}}{R_a} + \frac{L}{R_p}$: Figure 3.15 (c) illustrates an example case in which $R_p \leq R_a \leq 2R_p$. Therefore, by employing two processors in parallel, with each having sufficient memory to hold two input data-blocks as well as the partitioning information for one processed data-block, we obtain a response time of,

$$\tau_{resp} = \frac{L - N_0}{R_a} + \frac{L}{(P \times R_p)} \tag{3.9}$$

where $P = 2$. Accounting for P, this is identical to scenario (1). However,

similar to scenario (1-1), it is not feasible to obtain a valid τ^{α}_{sett}.

– **Scenario (2-2):** $\frac{1}{R_c} \geq \frac{B^{\alpha}_L}{R_a} + \frac{L}{R_p}$: The response time τ_{resp} is identical to the case of Eq. 3.9. The settling time can be obtained similar to Eq. 3.8 as,

$$\tau^{\alpha}_{sett} = \frac{(B^{\alpha}_L + 1) \times L - N_0}{R_a} + \frac{L}{(P \times R_p)} \tag{3.10}$$

So, in order to satisfy the last design criterion for online density estimation in Section 3.1, when the underlying density estimation changes, a sliding window discards the older data-blocks, and the newly collected data-blocks gradually change the estimated density. By sustaining a higher processing rate relative to the arrival rate, the algorithm is always able to respond to a change, before the next change occurs. So, by including the past density estimations that are not outdated by sliding outside the window, (*i.e.* the last B^{α}_L blocks), the model is always up-to-date.

Therefore, it has been shown that the proposed method meets all of the six design metrics for online density estimation over stationary and non-stationary data streams.

The process for choosing the proper block size is similar to what was presented in Section 3.3.2. In streaming applications, the time constraints are mainly determined by the rate of arrival of data (R_a) and the processing rate (R_p), as discussed earlier. For non-stationary streams, the settling time τ_{sett} also comes into play, and the block size needs to be determined in a way that results in a settling time that is smaller

than minimum period of changes in the underlying density.

3.4.2.2 Simulations

In order to demonstrate application of the proposed algorithm in performing online density estimation, simulations are run for some synthetic and real datasets. Simulation results are presented and discussed in the following subsections.

3.4.2.2.1 Synthetic dataset with simple structure Simulation results for two examples with synthetic data streams are presented in this subsection. The data stream utilized in the first example is generated using the same 64-dimensional density functions as in Section 3.3. Simulation results are presented in Figure 3.16. The behavior of a non-stationary data stream is simulated by gradually changing the statistics over time. Specifically, the mean vectors corresponding to the joint density function for the first two dimensions are slightly shifted.

As Figure 3.16 shows, changing the statistics of the incoming data causes a rise in the KLD, and depending on the scenario, the estimator may or may not be able to return to the initial value of KLD, before occurrence of the next change. From Figure 3.6, with block sizes of $L = 10$k, 20k, and 40k, the numbers of required blocks, B_L^α to reach the target KLD of 0.3 are 8, 2, and 1, respectively. The block sizes and sliding window sizes are denoted in the legends of the figure (b and W).

The plots with dashed lines show the results obtained from weighted averaging, which will be discussed in the next subsection.

The changes in the underlying density function are induced in various rates, to demonstrate different scenarios. It should be noted that changes in the density are not necessarily aligned with the boundaries of the blocks; so there are cases where a single block of data spans across two different underlying densities. From inspection of the results, it can be seen that in Regions 1, 3, and 7, there is enough time for the estimator to reach the target KLD, with any of the block sizes. However, this is not the case in other regions, where the rates of change in the statistics are higher.

For instance, consider the the case of $L = 10\text{k}$ with $B_{10\text{k}}^{0.3} = 8$. In this case, the response time τ_{resp} which is the time needed to collect and process one 10k block of data is short. But the settling time τ_{resp} can not be reached because the next change in the statistics occurs before eight blocks are collected and processed. This corresponds to scenario (1-1) described before.

On the other hand, in the case of $L = 20\text{k}$, after each change, the KLD value settles back to its initial value. This is because $B_{20\text{k}}^{0.3} = 2$, and in all regions there is enough time to collect and process two blocks of data before the next change in the statistics. This case corresponds to scenario (1-2). For $L = 40\text{k}$, only one block is needed to achieve the target KLD of less than 0.3. Nevertheless, in Regions 4 and 6, the collected block contains the data span across the point of change, and the short duration between changes does not allow for collection and processing of a block with

the updated statistics. These cases again correspond to scenario (1-1).

The second synthetic stream example is a much higher dimensional (256D) data stream with different underlying statistics, where the first 64 dimensions have Gaussian distributions, and the remaining dimensions have bimodal Beta distributions [115]. Simulation results for this example are shown in Figure 3.17.

The block sizes and their corresponding averaging window sizes are determined based on an analysis similar to what was described for the example in Figure 3.16. Similarly, as long as the distance between two consecutive changes is larger than the processing time for blocks covered by the averaging window, the target KLD can be reached (Regions 2 and 4 in Figure 3.17). From equations 3.7 to 3.10, it can be surmised that for a given data arrival rate R_a, we can find a suitable choice of L, with certain block processing time (or equivalently parameters R_p and B_L^α) (Figure 3.6) and certain block memory usage (Figure 3.8), and a suitable choice of P to achieve the best response τ_{resp} and settling τ_{sett}^α times.

3.4.2.2.2 Weighted Averaging In the discussion so far, equal-weight averaging is assumed over the data-blocks, (*i.e.*, the arithmetic mean). However, for a non-stationary stream, weighted averaging might be a better choice. By assigning larger weights to the most recent blocks, we can produce a smoother response for the estimator. This idea is demonstrated in the plots with the dashed lines in Figure 3.16, where the averaging weights of {1/4, 3/4} and {1/64, 3/64, 5/64, 7/64, 9/64, 11/64,

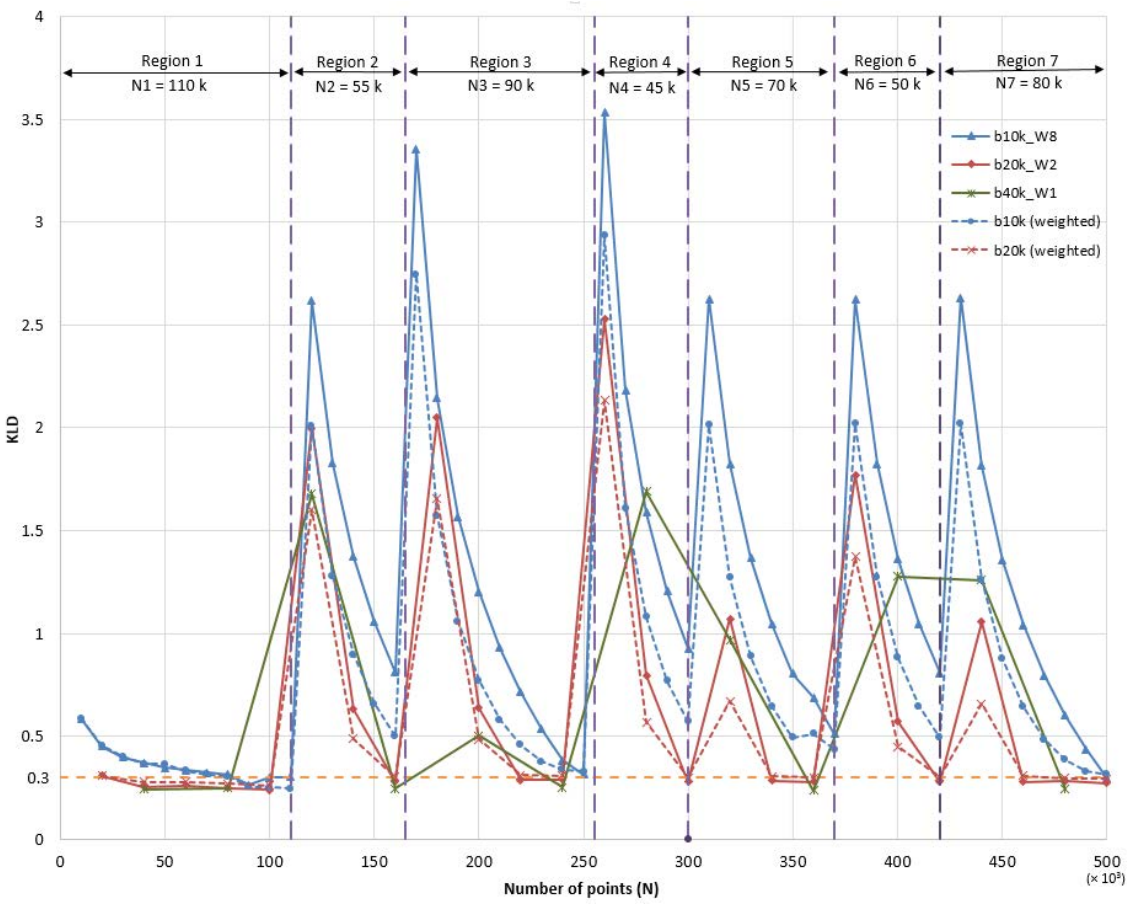


Figure 3.16: Variations in KLD, in density estimation over a 64-dimensional data stream.

13/64, 15/64} have been used for block sizes of 20k and 10k, respectively. While a more tailored made set of weighting factors are possible, a simple but natural set of weights (summed to unit) are chosen, which gradually increase from the oldest to the most recent block in a linear fashion.

As expected, in comparison with the case of an arithmetic mean, the cases with weighted averaging show a smaller surge in the value of KLD after each change in the statistics, and consequently, they reach the target KLD level faster.

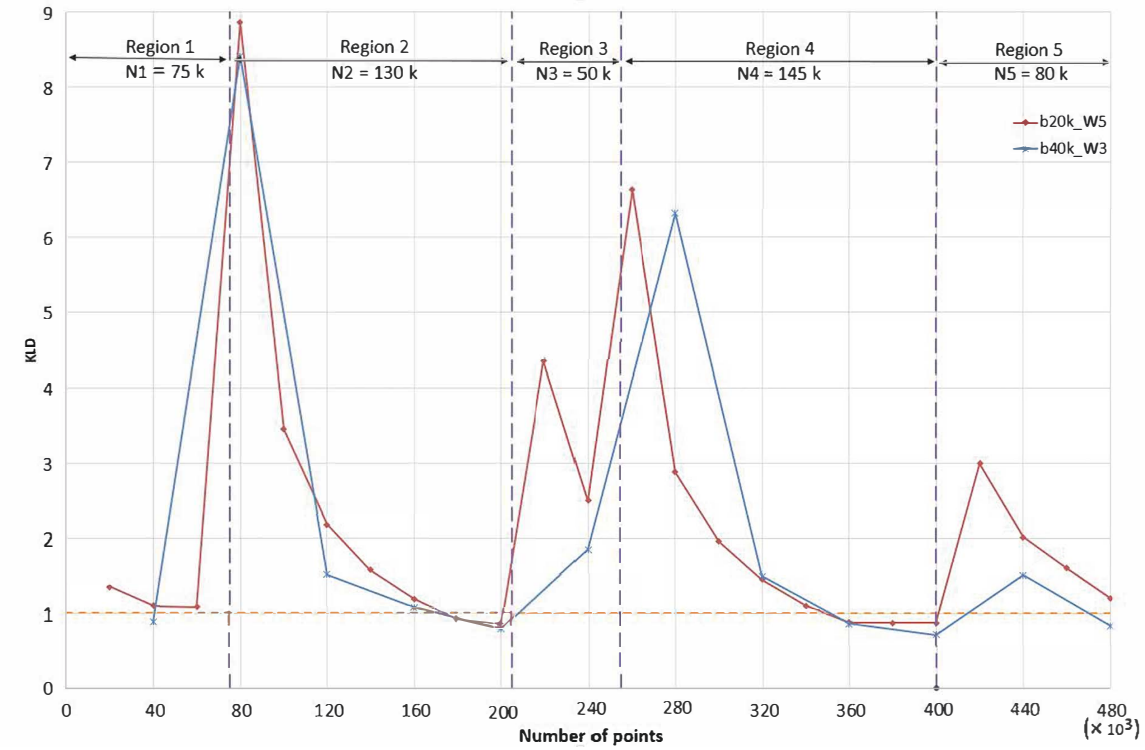


Figure 3.17: Variations in KLD, in density estimation over a 256-dimensional data stream.

3.4.2.2.3 Real dataset with complex structure Figure 3.18 presents the stream processing for a real complex structured 90-dimensional dataset. This is the same real dataset (called *MSD*) which was previously used in simulations of Section 3.3.2. To simulate the changes in the distribution, the dataset was sorted based on the release year and split into 3 intervals.

Since the true densities are not known and consequently the error between true and estimated densities is not available, the plots show the KLD between the block-averaged density estimation and the density estimates using the entire portion of the data, in each interval. The big jumps in the plots correspond to the locations in the dataset

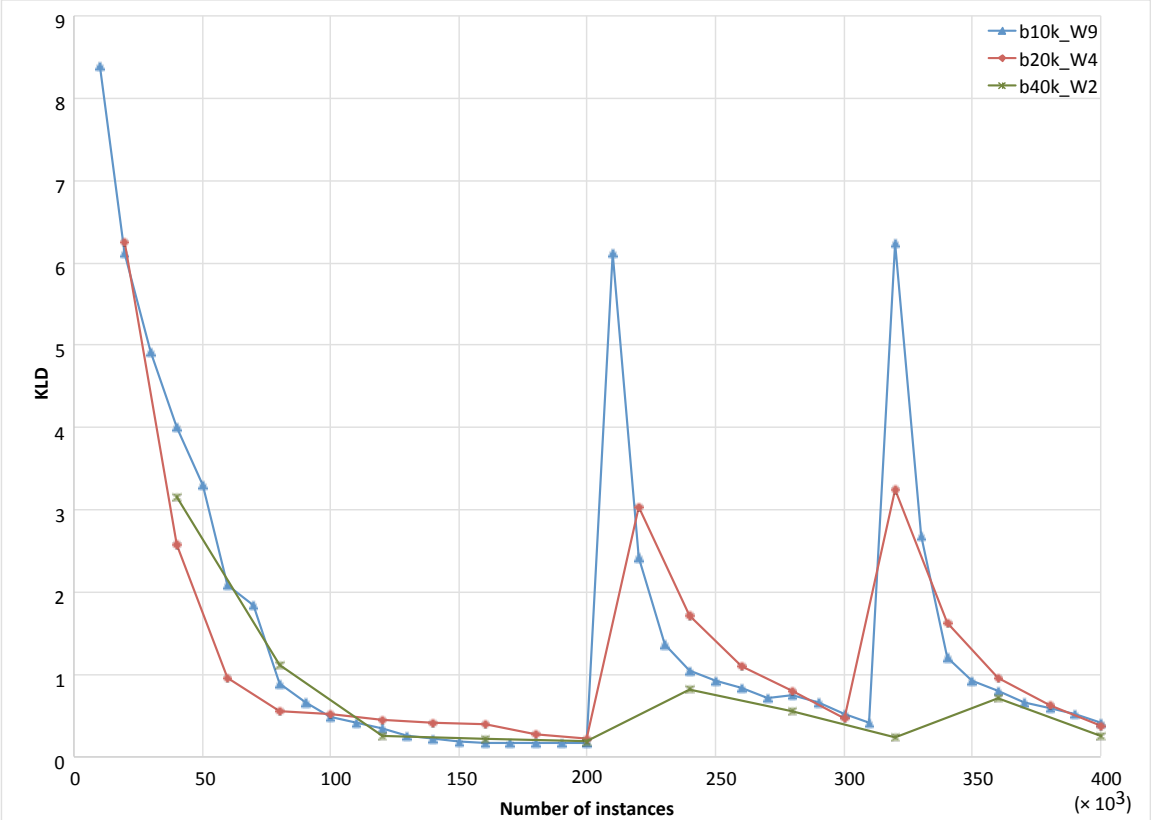


Figure 3.18: Variations in KLD, in density estimation over a 90-dimensional real data stream.

when there is a changes in the underlying density. The block sizes and averaging window sizes are determined based on the model previously developed in Section 3.3.2, in a way that the rate of change R_c is slow enough to allow for collection and processing of the required number of blocks to catch up with the change in data distribution and reach the target accuracy, before the occurrence of the next change. The pattern of density estimator response to the changes in data stream is similar to what was observed and discussed for the synthetic data in Figure 3.16.

3.5 Conclusions

A framework for density estimation over data streams was presented in this chapter. It uses a blockized implementation of the Bayesian sequential partitioning (BSP) algorithm. Performance analysis results were presented and discussed, to compare the performance of the regular BSP (with no blocking) with the blockized implementation, with various block sizes.

For datasets with simple data structures, a trade-off was observed between the estimation accuracy and the computation time on one side, and the memory requirements on the other. For datasets with complex structures, the computation time and the memory requirement fall on the same side of the trade-off.

The proposed algorithm for blockized BSP provides a suitable framework for online estimation of the data streams, as it successfully satisfies the general design criteria for systems with the mission of online mining of data over streams.

Since BSP is used as the density estimation core in this work, the proposed algorithm can be used for online density estimation over high-dimensional data streams with stationary or non-stationary statistics. Simulation results showed that for density estimation over a data stream with given R_a, R_p, and R_c, the optimum block size can be determined based on the underlying complexity of the data stream, and the system requirements with respect to the estimation error, computation time, memory usage and the number of available processors.

Chapter 4

Progressive Binary Partitioning

4.1 Introduction

In regular offline density estimation, an entire dataset is available for processing. Thus, the whole dataset is loaded into memory and processed to obtain an estimation of the density. In the previous chapter, online density estimation was discussed, to deal with situations where data is arriving as a stream, and needs to be processed in an online fashion. BBSP was introduced as a solution and its different aspects were discussed.

There are some other situations, where due to limitations on resources or performance restrictions of specific applications, processing the entire dataset in one run is not

possible. For instance,

- When the entire dataset it too large to fit in the memory.

- When enough memory space is available for loading the entire dataset, but processing the entire dataset of N samples of D-dimensional data takes a long time, while using only a subset of the dataset with $N_0 \ll N$ can give an estimate of almost equal accuracy in a much shorter time. The value of N_0 is of course not known to us in advance.

In this chapter, after a quick review of the BBSP method, an extension to this blockized method is proposed, which aims at improving the performance of this method by progressively partitioning the sample space as blocks of data are loaded and processed. This can have applications in both offline and online density estimation, as will be shown in the rest of this chapter.

Section 4.2 briefly goes through BBSP algorithm for offline and online density estimation. Section 4.3 proposes the method for progressive partitioning of the sample space, as an extension to the BBSP algorithm. Section 4.4 presents the simulation results for both offline and online cases. Finally, some concluding remarks are presented in Section 4.5.

4.2 BBSP Review

Let us consider a dataset of N samples and let us for now assume that the entire dataset is available, but due to time or memory restrictions, processing the entire dataset in one run is not an option. The alternative approach, BBSP, is based on the idea of dividing the dataset into chunks or *blocks*, processing each block separately, and taking the average of the block-wise estimates.

Here are the different steps of BBSP for a dataset of size N and sample space Ω:

1. The sample space Ω is divided into B data-blocks of equal sizes. Each block b, containing $L = N/B$ data instances, has a sample space, $\Omega^{(b)}(b = 1, .., B)$, which can be considered as an approximation of the entire sample space.

2. Each block is processed independently, using BSP algorithm. The results from all blocks are in fact B sets of subregion coordinates and their corresponding data counts. For each block, only these information about the partition map need to be stored, and the data block itself can be discarded to free the memory space for the next block to be loaded.

3. Now, for a given test point $\mathbf{z}$, B block-wise estimates of its density, $\hat{f}_b(\mathbf{z})$ $(b = 1, \ldots, B)$, can be obtained by mapping point $\mathbf{z}$ onto each of the B partitions to find its corresponding subregion in each sample space. The density in each

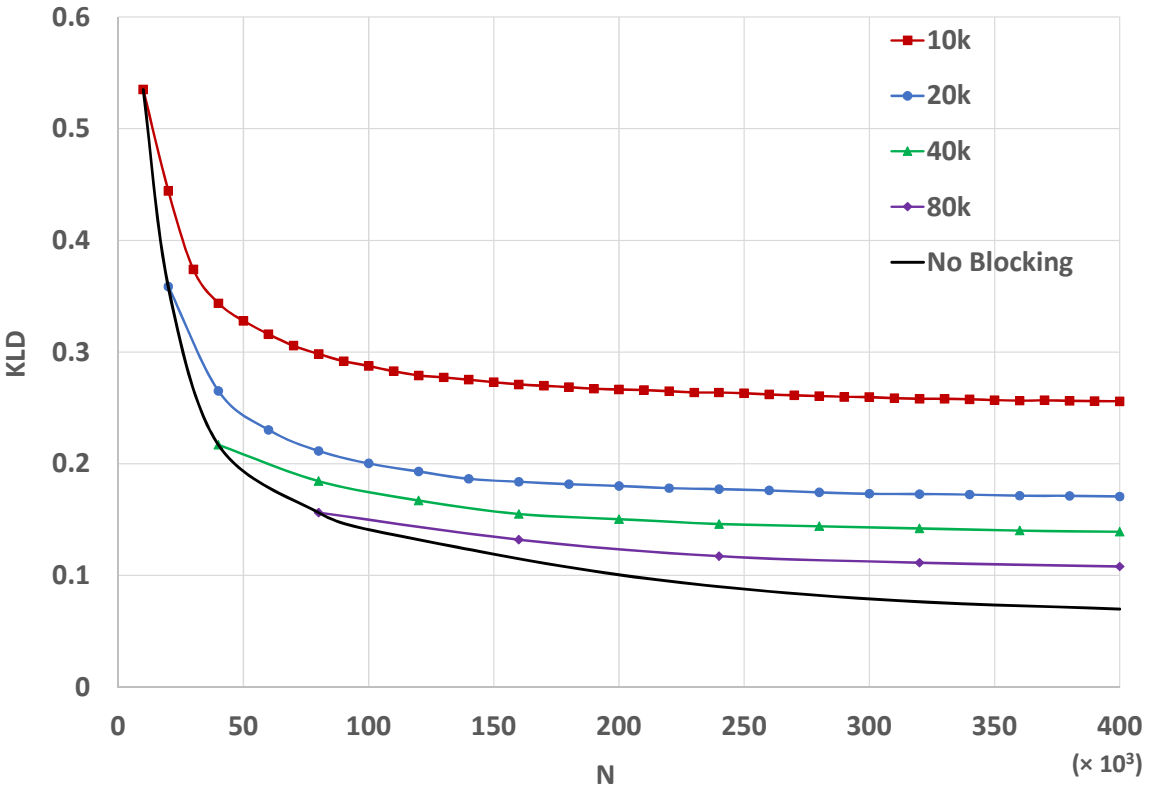


Figure 4.1: Density estimation error vs. N, for a range of block sizes.

subregion is estimated by calculating the normalized ratio of the data count in the subregion to its volume.

4. At the end, the final value for the estimated density $\hat{f}(\mathbf{z})$ is obtained by taking the average of the block-wise estimated densities $\hat{f}_b(\mathbf{z})$ over all B blocks.

$$\hat{f}(\mathbf{z}) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\mathbf{z}) \tag{4.1}$$

Figure 4.1 shows how the estimation error decreases with increasing the sample size. The dataset used in this example is a 64-D synthetic dataset with $N = 400,000$ samples. This dataset is going to be used in simulations in next sections for the

purpose of performance evaluations. The case with regular offline algorithm is compared with blockized approach, for a range of block sizes form $L = 10k$ to $L = 80k$. Kullback-Leibler divergence (KLD) [41] between the true density and estimated density is used as a measure of the estimation error. It can be seen that all plots reach a saturation point after which adding more samples does not significantly improve the estimation accuracy. The plots also show that by increasing the block size, the estimation accuracy of the blockized approach can be made arbitrarily close to that of the offline (non-blocking) approach.

Now this blockized model can be used for both offline and online density estimation purposes.

In offline density estimation, the block-wise estimates obtained from all of the previously processed blocks of data are used in the block averaging process, but in online cases with non-stationary streams, there needs to be an averaging window of certain size W, which only covers the last W blocks of data, such that only the most recent blocks of data are used. Clearly, for a real application the points of changes in the density are not known in advance. Therefore, in order to gradually adopt to the changes in the density, my online density estimator replaces the averaging method in Eq. 4.1 with a weighted block averaging method for online non-stationary streams. The weighted averaging approach assigns highest weight to the most recent block of data and lower weights to the older blocks in the window.

4.3 Progressive partitioning algorithm

This section presents an extension to the BBSP algorithm described in Section 4.2, with the purpose of improving the performance of the algorithm, in terms of processing time, for applications with strict performance requirements and limited computing power.

For example, let us consider an online density estimator: assume data is arriving at a constant rate, in an open-ended stream. The blockized density estimation algorithm described in the previous section can be used to provide an up-to-date estimate of the underlying density function at any time. As discussed before, for a given arrival rate, the processing unit needs to be able to finish processing each block, before the next block is fully collected. Otherwise, the estimator will eventually fall behind and will not be able to provide the most up-to-date estimate. One way to tackle this issue would be increasing the processing power, either by using a more powerful processor, or by adding more processing units to deploy parallel processing of blocks. Alternatively, we can try to make processing each block of data computationally less expensive.

Here *progressive partitioning* of the sample space is proposed as a solution. The basic idea of progressive partitioning is saving computation time by progressively updating the sample space partitions, using the newly received block of data. Starting with

the first block of data, the BSP algorithm is applied to partition the sample space and store the subregion coordinates and their corresponding data counts.

For the second block and beyond, instead of starting the partitioning process from beginning, i.e. the entire non-partitioned sample space, the algorithm uses the existing partitions from the previous block(s) and adds more cuts, as needed.

This will be described in more detail later in this section. The main question to ask at this stage is that for the second block and beyond, how does the algorithm decide which subregion to pick for making more cuts?

Similar to the regular BSP algorithm, this can be decided based on the distribution of data in subregions. Recall from the description of the BSP algorithm in Chapter 2 and the 2-dimensional example presented in Figure 2.2 that the subregion in which data is less uniformly distributed has the highest chance for being picked for the next cut. At level j of the sequential partitioning algorithm, the sample space has already been divided into $j-1$ subregions and a decision needs to be made on which subregion to cut next (cut_j). It is evident that for a D-dimensional sample space with $j-1$ subregions, there are $(j-1) \times D$ possibilities for the next cut. For each one of these possibilities, a conditional probability P_j is defined as follows [9]:

$$P_j(cut_j|g_{j-1}) = C_j 2^{n_k} \frac{\Gamma(n_k^{(1)})\Gamma(n_k^{(2)})}{\Gamma(n_k)} \tag{4.2}$$

where n_k is the total number of data points in existing subregion k, and $n_k^{(1)}$ and $n_k^{(2)}$ show how the data points would be split between the two new smaller subregions, as a result of making cut_j in subregion k. Here $\Gamma(.)$ denotes the Gamma function, C_j is the normalization constant, and $n_k = n_k^{(1)} + n_k^{(2)}$.

The conditional probability is defined in a way that P_j is large when $n_k^{(1)}$ and $n_k^{(2)}$ have a big difference. This means that the potential cuts that divide an existing subregion into two largely unbalanced subregions have the highest chance of being picked for the next cut.

Now, assuming that the dataset has been split into B blocks of equal size, the algorithm for progressive partitioning can be summarized as follows:

- *Step 1*: Start with the first block of data. Deploy the regular BSP algorithm to sequentially and adaptively cut the sample space. Let us assume that by processing the first block, a total of J_1 subregions are created in the sample space. At the end of this step, each of the samples in the first block is assigned to one of the subregions p_1 to p_{J_1}. Coordinates of all subregions p_j $(j = 1, \ldots, J_1)$ and their corresponding data counts are saved for use in the next step.

- *Step 2*: Discard the previous block of data and load the current block for processing.

- *Step 3*: Use the partition coordinates from the previous block to assign each of

the data instances of the current block to one of the existing subregions.

- *Step 4*: Count the number of data in each subregion.

- *Step 5*: Use the data count in each subregion and Eq. 4.2 to compute the corresponding conditional probabilities P_j for each possible cut.

- *Step 6*: Use the created probability mass function to randomly decide the next subregion to be cut.

- *Step 7*: Continue making sequential cuts, using the basic BSP algorithm, until all required cuts for the current block are made.

- *Step 8*: Repeat steps (2) to (7) until all blocks of data are processed.

At the end of the above process, or any time during the process, for any given point $\mathbf{z}$, an estimate of the density can be obtained by taking the average of the block-wise estimates obtained so far (Eq. 3.1).

This method of progressively updating the partitions is expected to save computation time by re-using the information obtained from processing previous blocks of data and reducing the number of binary cuts made for each block (except for the first block).

Quantitative evaluations of the effectiveness of the progressive partitioning method will be presented in the next section.

4.4 Simulations

This section presents the simulation results to evaluate the performance of the proposed progressive partitioning algorithm in offline and online density estimation. The progressive approach will be compared to the regular block averaging method in terms of estimation accuracy and computation time.

4.4.1 Offline example

For the offline case, it is assumed that the entire dataset is collected and available for use, but, as discussed earlier in Section 4.2, it is either impossible or undesirable to load and process the whole dataset in one run. For offline simulations, the synthetic dataset introduced in Section 4.2 is used. As stated earlier, deploying progressive partitioning approach mainly aims at reducing the computation time. Therefore, the computation times for the two methods are compared. However, it is also essential to make sure that the speed-up is not obtained at the cost of significant loss of estimation accuracy.

Plots in Figure 4.2 present the estimation error for a range of block sizes ($L = 10k$, $L = 20k$, $L = 40k$). Each plot shows how the KL divergence between the estimated

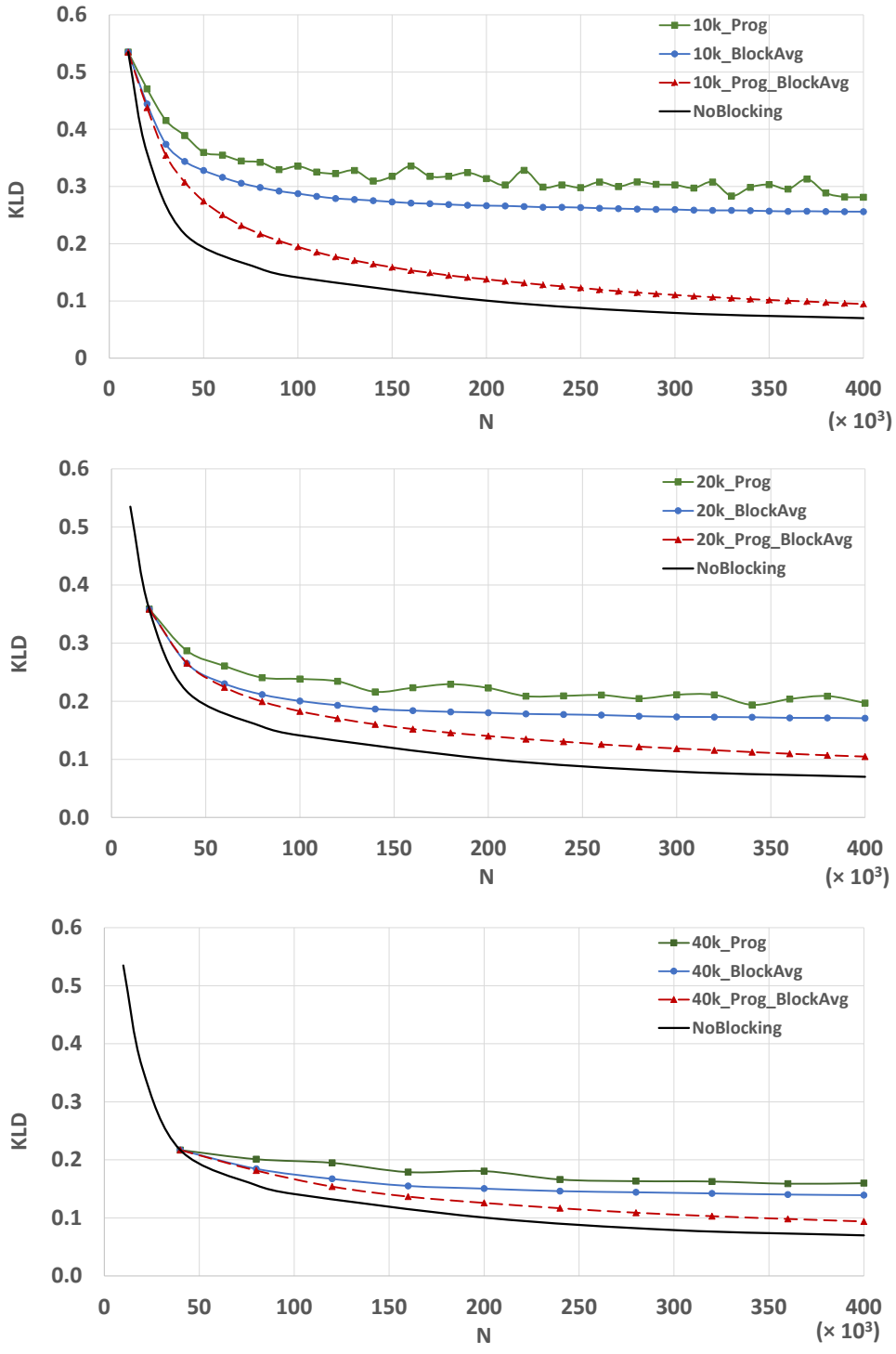


Figure 4.2: Comparison of methods in terms of KLD vs. N, for various block sizes (64-D data).

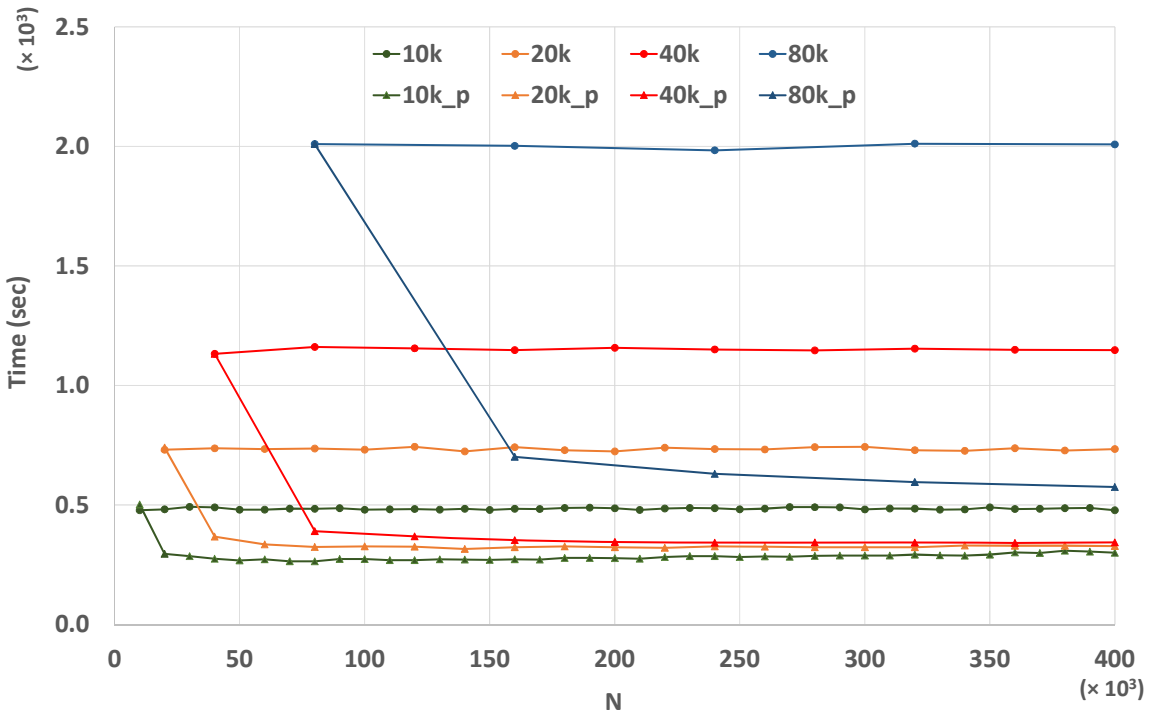


Figure 4.3: Comparing the methods of regular and progressive block averaging, regarding computation time per block of data, for various block sizes.

density and the true density changes as the number of processed samples varies from $N = 10k$ to $N = 400k$. KLD values associated with 4 different combinations of the possible estimation methods are compared:

- *No Blocking*: This is the original implementation of the BSP algorithm, where the entire subset of the available data is processed in one run, with no blocking.

- *Block Averaging*: The original form of blockized BSP (BBSP), as described in Section 4.2.

- *Progressive Block Averaging*: The progressive partitioning approach, with

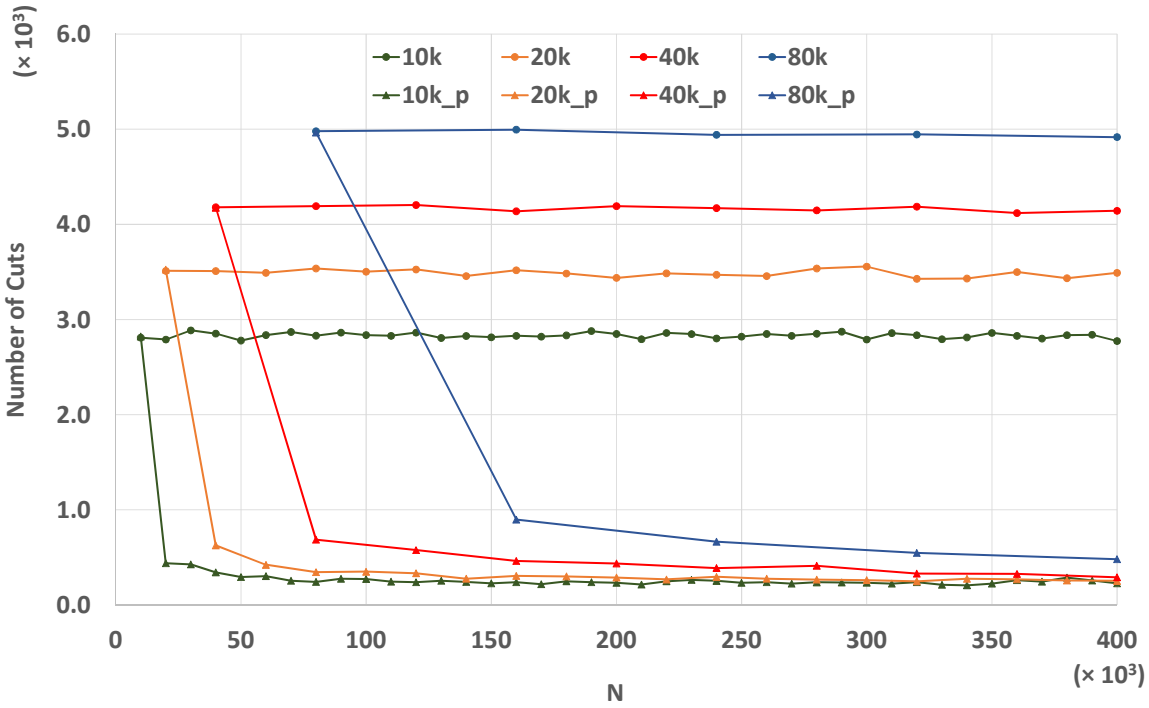


Figure 4.4: Comparing the methods of regular and progressive block averaging, regarding the number of binary cuts per block of data, for various block sizes.

weighted block averaging, as described in Section 4.3.

- *Progressive*: The progressive partitioning approach, excluding the weighted block averaging.

It can be seen in Figure 4.2 that for all 3 block sizes, progressive partitioning with block averaging leads into reduced estimation error. This can be explained by the fact that in the regular block averaging method, each block-wise estimate of the density is obtained using only the data in the current block, while in progressive approach, each block of data uses the partitioning created by the previous blocks as the starting point and adds more cuts to it; thus, in a sense, it is including the information from

all of the previously processed blocks of data.

On the other hand, the progressive partitioning approach with no block averaging has a slightly higher estimation error, compared to the block averaging approach, and it has a less smooth curve.

Another observation is that, as the block size increases, the KLD curves for all four approaches get closer to eachother. This is due to the fact that with larger block sizes, at any point, each single block of data contains a larger amount of information about the underlying density function. Thus trying to include the information obtained from the previous blocks will not lead into a significant improvement in the accuracy of the density estimation.

Also, as expected, for all three block sizes, the curves associated with no blocking lie underneath all the other curves. This means that in offline applications, if available resources allow, the best choice is to load and process the entire dataset in one run.

Figure 4.3 shows the computation time per block of data for the same range of block sizes, plus an additional case of 80k, for demonstration purposes. For each block size, two curves are showing computation time per block, corresponding to the basic block averaging approach, as well as the progressive partitioning with block averaging. The case of progressive partitioning without block averaging is not shown as a separate case, because the processing times for this case are almost same as those of the progressive case with block averaging. This is because the time associated

with the block-averaging part of the algorithm is an insignificant fraction of the block processing time.

Each pair of curves for each block size start at a common point, because in both approaches, processing the first block of data follows the same procedure, and the difference between non-progressive and progressive methods start from the second block of data. For the second block and beyond, the progressive approach saves a great deal of computation time, because it has to make fewer number of cuts, compared to the regular BBSP. Instead of starting from scratch, it picks up where the previous block finished and makes a few additional cuts using the newly processed block of data.

Figure 4.4 shows how the number of cuts significantly drops after the first block of data. For the range of block sizes shown in this figure, the number of cuts per block for the progressive partitioning cases are on average 5 to 6 times lower than the respective non-progressive approach.

As the results show, the progressive case without block averaging is inferior to the other cases in terms of estimation accuracy, and since its computation time is similar to the progressive case with block averaging, in the simulations for the online case in next subsection, the results for this case will not be presented.

4.4.2 Online example

In online density estimation, the case of stationary data streams is very similar to the offline case, because in both cases the underlying density function does not change across the blocks of data. The only difference is that in the online case, the data needs to be collected from a stream. Therefore, this section is only focusing on the more general case of non-stationary streams, where the distribution of data changes over time.

For the simulations in this section a 64-dimensional synthetic data is used, similar to what was used in simulations for the offline case. The changes are artificially generated by slightly modifying the statistics of the density function that is used to generate the random sequence of data.

Figure 4.5 shows the simulation results for online density estimation over the first 500,000 samples of this non-stationary stream. The dashed vertical lines locate the points at which the changes in the underlying take place. The plots show the variations in KL divergence between the true and estimated densities, for a range of block sizes. For each block size, the KLD values are compared for the regular BBSP approach and the progressive partitioning approach with block averaging.

In simulations, averaging windows of sizes 8, 2, and 1 are used for block sizes of

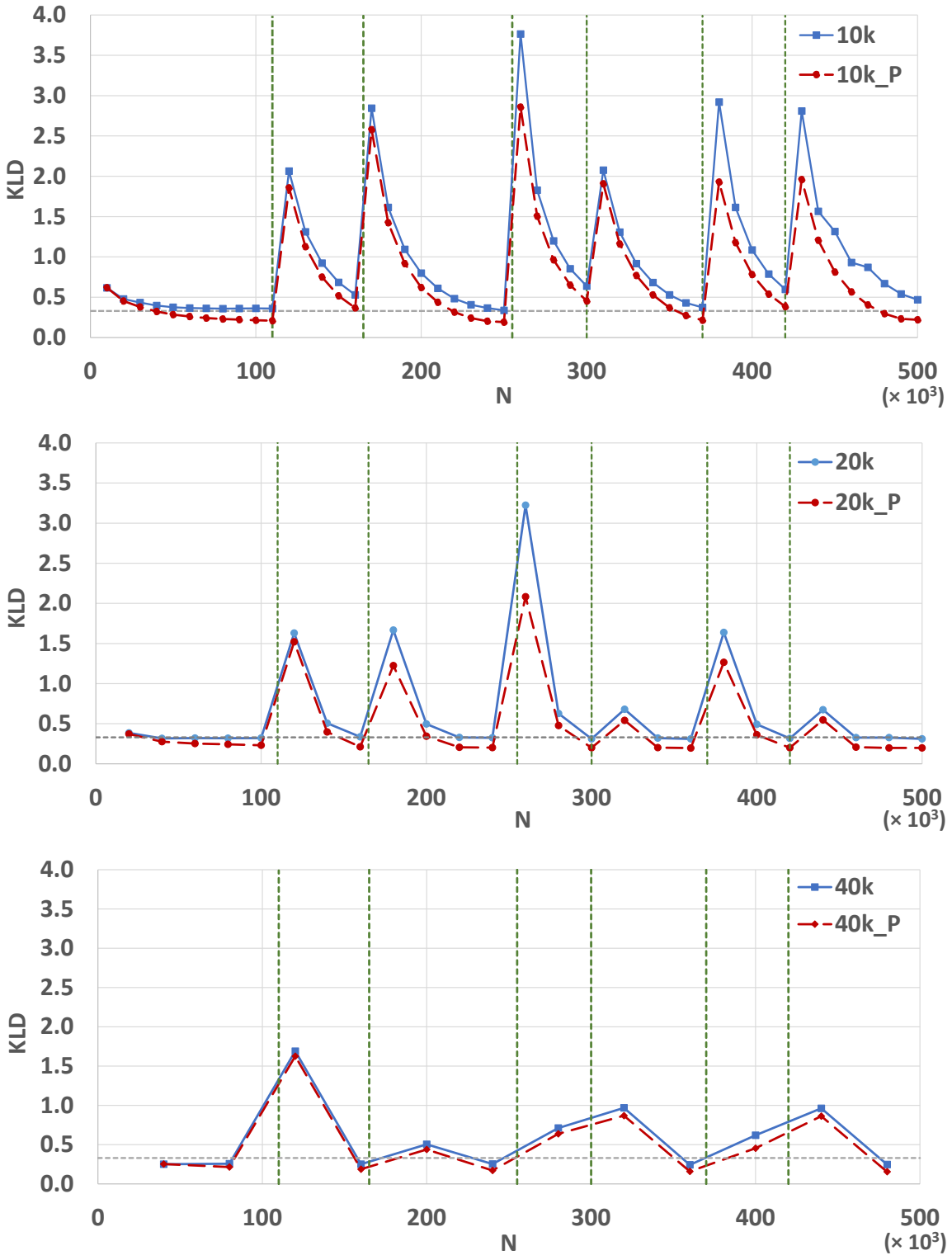


Figure 4.5: Comparison of estimation error, for various block sizes, for a stream of 64-D data.

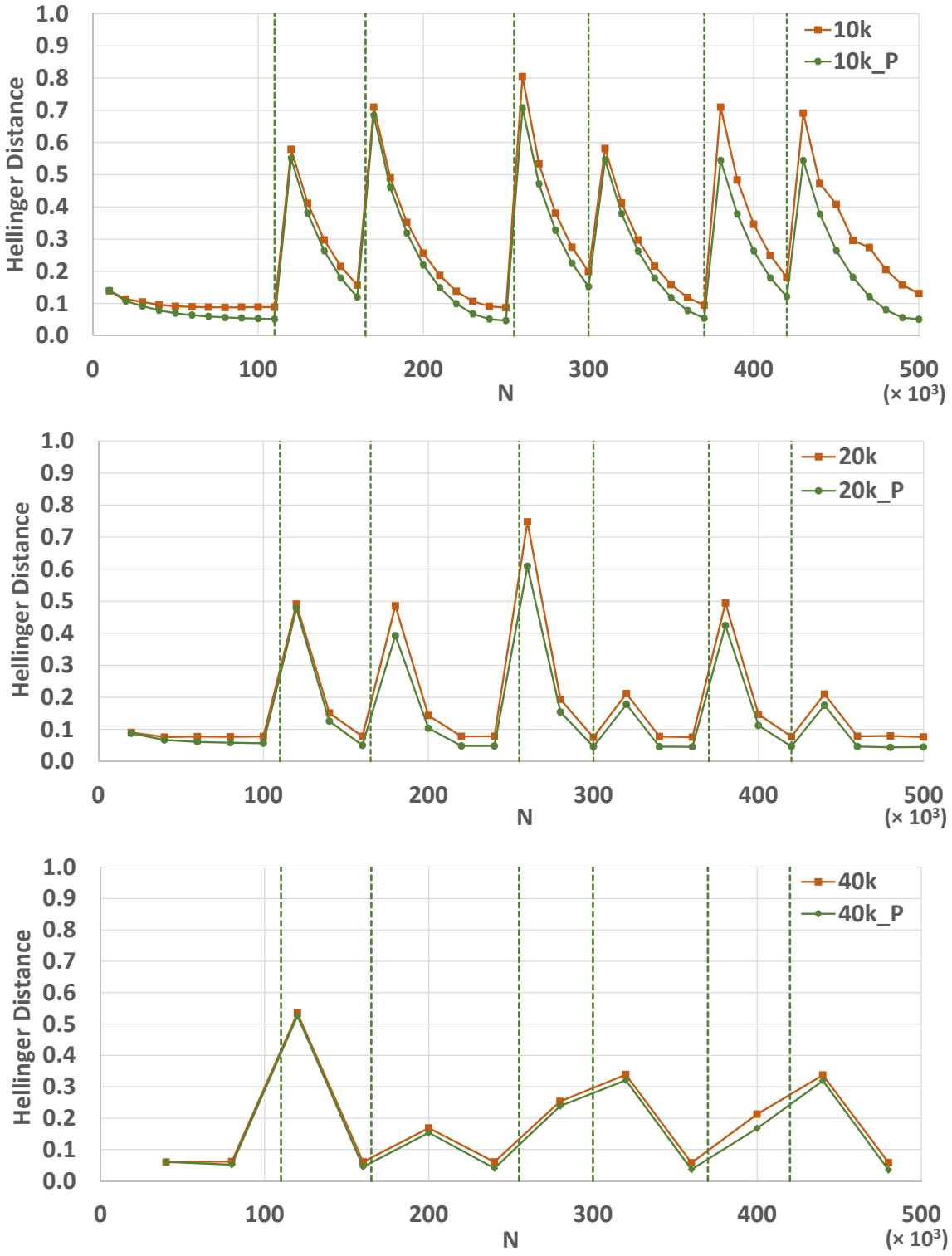


Figure 4.6: Comparison of Hellinger distance, for various block sizes, for a stream of 64-D data.

10k, 20k and 40k, respectively. These window sizes are chosen based on the results obtained from the offline simulations in previous subsection, to achieve certain level of estimation accuracy, in this case $KLD \leq 0.3$, shown in horizontal dashed line in Figure 4.5. For current online simulations, the set of weighting factors used for 10k and 20k cases are {1/64, 3/64, 5/64, 7/64, 9/64, 11/64, 13/64, 15/64} and {1/4, 3/4}, respectively. In the block averaging process, this weighting scheme gives highest wight to the density estimates obtained from the most recent block of data, while the older blocks will gradually fade out and be removed from the averaging process.

As the results presented in Figure 4.5 show, for all block sizes, using progressive partitioning reduces the KL divergence in the entire range of N. The plots show a jump in KL divergence, right after each change in the density (the vertical dashed lines). This is because after each change, both density estimation methods are still using some block-wise densities obtained from the blocks that were collected before the occurrence of the change, i.e. form the outdated data. However, as the plots show, with progressive partitioning approach the jump in the KLD is smaller than the other method. Also, after each change, the progressive approach reaches the target level of KLD faster than the non-progressive approach.

For this stream, the maximum rate of change of density is occurring when two consecutive changes (the third and fourth one) are separated by 45,000 samples. Thus, considering the window sizes mentioned above, the proper choice for the block size

would be 20k. With this block size, as the plots show, in all 7 regions shown in the plots, the estimator is able to collect and process enough number of blocks of data (2 blocks) to reach the target KLD, before the next change in the density occurs. With 10k blocks, there are several cases in which the distance between two consecutive changes is less than arrival time for 80k data point, and thus the distribution of data changes before 8 blocks can be collected. A similar situation holds for the case of 40k blocks.

It is also observed that as the block size is increased, the difference in KLDs associated to the progressive and non-progressive approach become less significant. This is similar to the observation made in offline simulation results.

As an additional measure of performance, Figure 4.6 is presenting the online density estimation error in terms of Hellinger distance [15] between the true density and the estimated density. It can be seen that the variations in Hellinger distance are very similar to KLD plots shown in Figure 4.5.

A comparison of the computation times is not presented, as they are similar to what was presented in the previous subsection for offline case. The only difference is that in online (non-stationary) case, the saving made in computation time is more useful, compared to the offline case. It will make it possible for the online density estimator to provide a faster updated estimate of the density after each change in the distribution of the data. In terms of the quantitative measures introduced in Section 3.4.2, use

of progressive partitioning method reduces the response time and settling time in non-stationary streams.

Thus, the simulation results presented in this section showed the effectiveness of the proposed progressive partitioning approach in online density estimation over non-stationary streams.

4.5 Conclusions

In this chapter, a method for improved density estimation for high-dimensional data was presented, for situations where processing the entire dataset in one run is either impossible or not an optimum choice. An extension to the BBSP method was proposed for improving the performance by progressively updating the sample space partitions. For each block, the sample space partition created from the previously processed block(s) is used as the starting point, and more cuts are made in the sample space. This method provides a means for utilizing the information from all the processed blocks, as long as they are inside the averaging window.

Matlab simulations were performed to evaluate the effectiveness of the proposed progressive partitioning algorithm in offline and online high-dimensional density estimation. The simulations compared the basic block-wise averaging method with

progressive partitioning approach, for a range of block sizes.

In offline cases, simulation results show an improvement in both computational efficiency and estimation accuracy. That is, using the progressive partitioning approach leads into obtaining more accurate estimation in a shorter time, which can be useful in efficient processing of Big data.

In online cases with non-stationary streams of data, if progressive partitioning is used, after each change in the underlying density, the raise in KLD will be lower and it will have shorter response time and settling time compared to the regular weighted block averaging method. These improvements will be helpful in applications with limited computing resources or strict timing requirements.

Chapter 5

Conclusions and Future Works

5.1 Summary

In this dissertation, computationally efficient offline and online density estimation for high-dimensional data was presented. Chapter 1 provided an introduction to the subject and motivation of the work. In Chapter 2, a framework for high-dimensional density estimation was discussed, which is based on Bayesian sequential partitioning of the sample space, and use of copula transform for reducing estimation error and computation time. Also, some example applications in density-based classification and clustering were presented. Chapter 3 discussed a blockized density estimation method, BBSP, for fast and accurate estimation of densities in both offline and online

applications. Chapter 4 presented an extension to the BBSP algorithm, in which partitions created from one block are used in the following blocks as the starting point for progressive partitioning of the sample space. This progressive partitioning approach was shown to improve the estimation accuracy as well as computation time in both offline and online density estimation problems.

5.2 Suggestions for Future Works

In the following, there are a few ideas for future works in the area of online density estimation and data mining, based on what was presented in this dissertation.

- **Online density-based classification and clustering for high dimensional data**

  Using the online density estimation framework developed in my Chapter 3, online density-based classification and clustering methods can be developed for high-dimensional data over stationary and non-stationary streams. In online classification, for streams of data, the training stage can be done in real-time, using the blockized density estimation method developed in this dissertation.

- **Use of density estimation in change detection**

  In dealing with real (non-synthetic) non-stationary data streams, the points of

occurrence of abrupt changes are of course unknown. Automated detection of these changes will make it possible to significantly improve the performance of the online density estimation, by dynamically assigning the averaging weights, as explained in the third item below.

The change detection algorithm is based on the idea of a multi-resolution density estimation algorithm using the blockized density estimation method developed in Chapter 3.

In the extended version of the algorithm, there will be two different density estimators working in parallel: the first one (main estimator) uses the optimum block size calculated for the specific application; the second one (change detector) would use a block size which is a fraction the full block size. The change detector aims at continuously obtaining a quick low-resolution estimate of the underlying density. The low-resolution estimates from consecutive blocks can be compared in terms of some difference or divergence measure (like Kullback-Leibler divergence or Hellinger distance) to detect abrupt changes. A sudden big rise in the differences between consecutive blocks can be considered as a sign of an abrupt change in the underlying probability density function.

Using copula transform, change detection can be performed independently, over each marginal. If resources are available, parallel processing can be employed to accelerate this process.

- **Online density estimation with dynamic averaging weights**

  Using the change detection algorithm described in the second item above, an improvement can be made to the online density estimation method. In the algorithm described in Chapter 3, the data (arriving over a stream) is collected and processed in chunks or blocks of fixed size. Each block of data is processed separately, and block-wise estimates are obtained. The overall estimate of the underlying density function is then obtained by taking a weighted average of the block-wise estimates. Also, the size of the sliding averaging window and weight factors are constant.

  For non-stationary streams, performance of the online density estimator can be improved by changing the averaging weights upon detection of abrupt changes in the underlying distribution of the data. In my blockized density estimation approach, higher weights are assigned to the most recent blocks of data, which makes it possible for the estimator to smoothly adapt to gradual changes in the underlying distribution, without noise-like oscillations. However, in case of abrupt changes in the underlying distribution, it would be more suitable to assign a weight of 1 to the most recent block and 0 for all the older blocks, such that only the data from updated distribution is used in the density estimation process. This is in fact equivalent to resetting the averaging window size to 1, upon detection of each abrupt change, and then increasing it gradually as more blocks of updated data are collected.

References

[1] P. Simon, *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013.

[2] C. Heinz and B. Seeger, "Cluster kernels: Resource-aware kernel density estimators over streaming data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, pp. 880–893, July 2008.

[3] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[4] M. Bressan and J. Vitria, "Nonparametric discriminant analysis and nearest neighbor classification," *Pattern Recognition Letters*, vol. 24, pp. 2743–2749, November 2003.

[5] Y. Motai and H. Yoshida, "Principal composite kernel feature analysis: Data-dependent kernel approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 1863–1875, August 2013.

[6] E. Müller, I. Assent, R. Krieger, S. Günnemann, and T. Seidl, "Densest: Density estimation for data mining in high dimensional spaces," in *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp. 175–186, SIAM, 2009.

[7] J. H. Friedman and N. I. Fisher, "Bump hunting in high dimensional data," *Statistics and Computing*, vol. 9, pp. 123–143, April 1999.

[8] K. Wu, K. Zhang, W. Fan, A. Edwards, and S. Y. Philip, "Rs-forest: A rapid density estimator for streaming anomaly detection," in *Data Mining (ICDM), 2014 IEEE International Conference on*, pp. 600–609, IEEE, 2014.

[9] L. Lu, H. Jiang, and W. H. Wong, "Multivariate density estimation by bayesian sequential partitioning," *Journal of the American Statistical Association*, vol. 108, pp. 1402–1410, December 2013.

[10] K. Yang and W. H. Wong, "Discovering and visualizing hierarchy in multivariate data," *arXiv preprint arXiv:1403.4370*, 2014.

[11] B. M. Wise and P. Geladi, "A brief introduction to multivariate image analysis (mia)," *Eigenvector Research, Inc., http://www. eigenvector. com/Docs/MIA_Intro. pdf*, 2000.

[12] L. E. Mujica, J. Vehí, M. Ruiz, M. Verleysen, W. Staszewski, and K. Worden,

"Multivariate statistics process control for dimensionality reduction in structural assessment," *Mechanical Systems and Signal Processing*, vol. 22, no. 1, pp. 155–171, 2008.

[13] T. W. Nattkemper, "Multivariate image analysis in biomedicine," *Journal of Biomedical Informatics*, vol. 37, no. 5, pp. 380–391, 2004.

[14] A. O. Kirdar, K. D. Green, and A. S. Rathore, "Application of multivariate data analysis for identification and successful resolution of a root cause for a bioprocessing application," *Biotechnology progress*, vol. 24, no. 3, pp. 720–726, 2008.

[15] D. W. Scott, *Multivariate Density Estimation, Theory, Practice and Visualization*. John Wiley & Sons, 1992.

[16] P. P. B. Eggermont and V. N. LaRiccia, *Maximum Penalized Likelihood Estimation, Volume I: Density Estimation; Springer Series in Statistics*. Springer, 2001.

[17] R. E. Bellman, *Dynamic Programming*. Dover Publications, 2003.

[18] D. L. Donoho *et al.*, "High-dimensional data analysis: The curses and blessings of dimensionality," *AMS Math Challenges Lecture*, vol. 1, p. 32, 2000.

[19] A. W. Bowman and A. Azzalini, *Applied Smoothing Techniques for Data Analysis*. Oxford University Press, 1997.

[20] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li, "Quality and efficiency for kernel density estimates in large data," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, (New York, NY, USA), pp. 433–444, ACM, 2013.

[21] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, "Improved fast gauss transform and efficient kernel density estimation," in *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 664–671 vol.1, Oct 2003.

[22] M. C. Jones, J. S. Marron, and S. J. Sheather, "Progress in data-based bandwidth selection for kernel density estimation," *Computational Statistics*, vol. 11, no. 3, pp. 337–381, 1996.

[23] S. J. Sheather and M. C. Jones, "A reliable data-based bandwidth selection method for kernel density estimation," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 683–690, 1991.

[24] D. Devroye, J. Beirlant, R. Cao, R. Fraiman, P. Hall, M. Jones, G. Lugosi, E. Mammen, J. Marron, C. Sánchez-Sellero, *et al.*, "Universal smoothing factor selection in density estimation: theory and practice," *Test*, vol. 6, no. 2, pp. 223–320, 1997.

[25] M. C. Jones, J. S. Marron, and S. J. Sheather, "A brief survey of bandwidth selection for density estimation," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 401–407, 1996.

[26] Z. I. Botev, J. F. Grotowski, and D. P. Kroese, "Kernel density estimation via diffusion," *Ann. Statist.*, vol. 38, pp. 2916–2957, 10 2010.

[27] P. Ram and A. G. Gray, "Density estimation trees," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 627–635, ACM, 2011.

[28] M. Vannucci, *Nonparametric density estimation using wavelets*. Institute of Statistics & Decision Sciences, Duke University, 1995.

[29] G. M. Araujo, F. M. Ribeiro, W. S. Júnior, E. A. da Silva, and S. K. Goldenstein, "Weak classifier for density estimation in eye localization and tracking," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3410–3424, 2017.

[30] A. Faroughi, R. Javidan, and M. Emami, "A new density estimator based on nearest and farthest neighbor," in *Telecommunications (IST), 2016 8th International Symposium on*, pp. 185–190, IEEE, 2016.

[31] J.-N. Hwang, S.-R. Lay, and A. Lippman, "Nonparametric multivariate density estimation: a comparative study," *IEEE Transactions on Signal Processing*, vol. 42, no. 10, pp. 2795–2810, 1994.

[32] D. W. Scott and S. R. Sain, "Multidimensional density estimation," *Handbook of statistics*, vol. 24, pp. 229–261, 2005.

[33] E. López-Rubio, "A histogram transform for probabilitydensity function estimation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 4, pp. 644–656, 2014.

[34] Y. Kawahara and M. Sugiyama, "Sequential change-point detection based on direct density-ratio estimation," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 5, no. 2, pp. 114–127, 2012.

[35] M. Sugiyama, T. Kanamori, T. Suzuki, M. C. d. Plessis, S. Liu, and I. Takeuchi, "Density-difference estimation," *Neural Computation*, vol. 25, no. 10, pp. 2734–2775, 2013.

[36] K. Yang, H. Su, and W. H. Wong, "co-bpm: a bayesian model for estimating divergence and distance of distributions," *arXiv preprint arXiv:1410.0726*, p. 13, 2014.

[37] M. Sugiyama, T. Suzuki, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanabe, "Direct importance estimation for covariate shift adaptation," *Annals of the Institute of Statistical Mathematics*, vol. 60, no. 4, pp. 699–746, 2008.

[38] X. Nguyen, M. J. Wainwright, and M. I. Jordan, "Estimating divergence functionals and the likelihood ratio by convex risk minimization," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5847–5861, 2010.

[39] W. H. Wong, L. Ma, *et al.*, "Optional pólya tree and bayesian inference," *The Annals of Statistics*, vol. 38, no. 3, pp. 1433–1459, 2010.

[40] T. S. Ferguson, "Prior distributions on spaces of probability measures," *The annals of statistics*, pp. 615–629, 1974.

[41] J. Klemelä, *Smoothing of Multivariate Data: Density Estimation and Visualization*. John Wiley & Sons, 2009.

[42] J. Klemelä, "Multivariate histograms with data-dependent partitions," *Statistica Sinica*, vol. 19, no. 1, pp. 159–176, 2009.

[43] G. Lugosi and A. Nobel, "Consistency of data-driven histogram methods for density estimation and classification," *The Annals of Statistics*, vol. 24, no. 2, pp. 687–706, 1996.

[44] J. Silva and S. S. Narayananl, "Information divergence estimation based on data-dependent partitions," *Journal of Statistical Planning and Inference*, vol. 140, pp. 3180–3198, 2010.

[45] A. J. Izenman, *Modern Multivariate Statistical techniques: Regression, Classification, and Manifold Learning*. Springer, 2003.

[46] J. S. Liu, *Monte Carlo Strategies in Scientific Computing; Springer Series in Statistics*. Springer, 2001.

[47] A. Kong, J. S. Liu, and W. H. Wong, "Sequential imputations and bayesian missing data problems," *Journal of the American Statistical Association*, vol. 89, pp. 278–288, March 1994.

[48] K. Yang, H. Su, and W. H. Wang, "Density estimation via discrepancy," *arXiv preprint arXiv:1509.06831*, 2015.

[49] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, vol. 63. Siam, 1992.

[50] T. D. E. Barat and T. Montagu, "Nonparametric bayesian inference in nuclear spectrometry," in *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, vol. 1, (Honolulu, HI), pp. 880–887, Oct 2007.

[51] D. W. Meyer, "Density estimation with distribution element trees," *Statistics and Computing*, vol. 28, pp. 609–632, May 2018.

[52] M. Sugiyama, S. Liu, M. C. Du Plessis, M. Yamanaka, M. Yamada, T. Suzuki, and T. Kanamori, "Direct divergence approximation between probability distributions and its applications in machine learning," *Journal of Computing Science and Engineering*, vol. 7, no. 2, pp. 99–111, 2013.

[53] T. Kanamori and M. Sugiyama, "Statistical analysis of distance estimators with density differences and density ratios," *Entropy*, vol. 16, no. 2, pp. 921–942, 2014.

[54] J. R. Hershey and P. A. Olsen, "Approximating the kullback leibler divergence between gaussian mixture models," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–317, IEEE, 2007.

[55] J.-L. Durrieu, J.-P. Thiran, and F. Kelly, "Lower and upper bounds for approximation of the kullback-leibler divergence between gaussian mixture models," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 4833–4836, Ieee, 2012.

[56] M. H. (ed.), *Encyclopedia of Mathematics*. Springer, 2001.

[57] A. Bayestehtashk and I. Shafran, "Parsimonious multivariate copula model for density estimation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (Vancouver, BC), pp. 5750–5754, May 2013.

[58] M.-S. Changa and X. Wub, "Transformation-based nonparametric estimation of multivariate densities," *Journal of Multivariate Analysis*, vol. 135, pp. 71 – 88, March 2015.

[59] A. Majdara and S. Nooshabadi, "Efficient data structures for density estimation for large high-dimensional data," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, May 2017.

[60] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[61] O. A. R. Board, "The OpenMP API specification for parallel programming." http://openmp.org/wp/, 2016.

[62] M. P. I. Forum, "MPI: A message-passing interface standard." http://www.mpi-forum.org/, 2016.

[63] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[64] A. C. Rencher and W. F. Christensen, *Methods of Multivariate Analysis*. Wiley, 2012.

[65] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[66] L. Rokach and O. Z. Maimon, *Data mining with decision trees: theory and applications*, vol. 69. World scientific, 2008.

[67] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[68] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[69] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, Sep 1995.

[70] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[71] I. Steinwart and A. Christmann, *Support vector machines*. Springer Science & Business Media, 2008.

[72] P. A. Lachenbruch and M. Goldstein, "Discriminant analysis," *Biometrics*, pp. 69–85, 1979.

[73] G. McLachlan, *Discriminant analysis and statistical pattern recognition*, vol. 544. John Wiley & Sons, 2004.

[74] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[75] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, pp. 41–46, IBM New York, 2001.

[76] D. Heck, J. N. Capdevielle, G. Schatz, T. Thouw, and F. K. Gmbh, "CORSIKA: A monte carlo code to simulate extensive air showers, report fzka 6019, forschungszentrum karlsruhe."

[77] M. Lichman, "UCI machine learning repository." http://archive.ics.uci.edu/ml, 2013.

[78] M. R. Anderberg, "Cluster analysis for applications," tech. rep., Office of the Assistant for Study Support Kirtland AFB N MEX, 1973.

[79] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, 2009.

[80] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, pp. 515–528, Mar. 2003.

[81] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[82] M. Steinbach, L. Ertoz, and V. Kumar, "Challenges of clustering high dimensional data. new vistas in statistical physics. applications in econophysics, bioinformatics, and pattern recognition," 2003.

[83] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[84] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[85] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "Density-based spatial clustering

of applications with noise," in *Int. Conf. Knowledge Discovery and Data Mining*, vol. 240, 1996.

[86] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[87] H.-P. Kriegel and M. Pfeifle, "Density-based clustering of uncertain data," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 672–677, ACM, 2005.

[88] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[89] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[90] P. F. et al, "Clustering basic benchmark," 2015.

[91] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[92] J. R. Vennam and S. Vadapalli, "Syndeca: A tool to generate synthetic datasets for evaluation of clustering algorithms.," in *COMAD*, pp. 27–36, Citeseer, 2005.

[93] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[94] M. P. Wand and M. C. Jones, *Kernel Smoothing*. Chapman & Hall/CRC, 1995.

[95] C. Heinz, *Density estimation over data streams*. PhD thesis, University of Marburg, 2007.

[96] M. Geilke, A. Karwath, and S. Kramer, "Online density estimation of heterogeneous data streams in higher dimensions," in *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*, ECML PKDD 2016, (New York, NY, USA), pp. 65–80, Springer-Verlag New York, Inc., 2016.

[97] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*. Cambridge university press, 2006.

[98] J. Walker, A. Gupta, and M. Hebert, "Patch to the future: Unsupervised visual prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3302–3309, 2014.

[99] A. Zhou, Z. Cai, L. Wei, and W. Qian, "M-kernel merging: Towards density estimation over data streams," in *Proc. of DASFAA*, pp. 285–292, 2003.

[100] N. G.-L. L. Zheng-Ding, "Density estimation over data stream," *Computer Science*, vol. 12, p. 025, 2006.

[101] C. Heinz and B. Seeger, "Cluster kernels: Resource-aware kernel density estimators over streaming data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, pp. 880–893, July 2008.

[102] C. Heinz and B. Seeger, "Resource-aware kernel density estimators over streaming data," in *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, (New York, NY, USA), pp. 870–871, ACM, 2006.

[103] V. A. Epanechnikov, "Non-parametric estimation of a multivariate probability density," *Theory of Probability & Its Applications*, vol. 14, no. 1, pp. 153–158, 1969.

[104] H. Shen and X. L. Yan, "Probability density estimation over evolving data streams using tilted parzen window," in *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pp. 585–589, July 2008.

[105] C. Heinz and B. Seeger, "Adaptive wavelet density estimators over data streams," in *Scientific and Statistical Database Management, 2007. SSBDM '07. 19th International Conference on*, pp. 35–35, July 2007.

[106] E. S. García-Treviño and J. A. Barria, "Online wavelet-based density estimation for non-stationary streaming data," *Computational Statistics & Data Analysis*, vol. 56, no. 2, pp. 327–344, 2012.

[107] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, (New York, NY, USA), pp. 71–80, ACM, 2000.

[108] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, (New York, NY, USA), pp. 97–106, ACM, 2001.

[109] P. Domingos and G. Hulten, "Learning from infinite data in finite time," in *Advances in Neural Information Processing Systems 14* (T. G. Dietterich, S. Becker, and Z. Ghahramani, eds.), pp. 673–680, MIT Press, 2002.

[110] M. Kristan, A. Leonardis, and D. Skočaj, "Multivariate online kernel density estimation with gaussian kernels," *Pattern Recognition*, vol. 44, no. 10-11, pp. 2630–2642, 2011.

[111] P. Domingos and G. Hulten, "Catching up with the data: Research issues in mining data streams," in *Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[112] V. Lemaire, C. Salperwyck, and A. Bondu, "A survey on supervised classification on data streams," in *Business Intelligence*, pp. 88–125, Springer, 2015.

[113] B. Hadjkacem, W. Ayedi, and M. Abid, "A comparison between person re-identification approaches," in *2016 Third International Conference on Artificial Intelligence and Pattern Recognition (AIPR)*, pp. 1–4, Sept 2016.

[114] A. Shaker, R. Senge, and E. Hüllermeier, "Evolving fuzzy pattern trees for binary classification on data streams," *Information Sciences*, vol. 220, pp. 34–45, 2013.

[115] S. Kotz, N. Balakrishnan, and N. Johnson, "Multivariate distributions volume 1: Models and applications," 2000.

[116] A. Zhou, Z. Cai, and L. Wei, "Density estimation over data stream," 2015.

[117] C. Heinz and B. Seeger, "Towards kernel density estimation over streaming data," in *COMAD*, 2006.

[118] C. Heinz and B. Seeger, "Exploring data streams with nonparametric estimators," in *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*, pp. 261–264, 2006.

[119] C. Heinz and B. Seeger, "Wavelet density estimators over data streams," in *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, (New York, NY, USA), pp. 578–579, ACM, 2005.

[120] P. Domingos and G. Hulten, "A general framework for mining massive data

streams," *Journal of Computational and Graphical Statistics*, vol. 12, no. 4, pp. 945–949, 2003.

[121] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," *SIGMOD Rec.*, vol. 34, pp. 18–26, June 2005.

[122] "Streaming-data algorithms for high-quality clustering," in *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, (Washington, DC, USA), pp. 685–, IEEE Computer Society, 2002.

[123] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Random sampling techniques for space efficient online computation of order statistics of large datasets," *SIGMOD Rec.*, vol. 28, pp. 251–262, June 1999.

[124] P. Van Kerm, "Adaptive kernel density estimation," *Stata Journal*, vol. 3, no. 2, pp. 148–156(9), 2003.

[125] G. Kollios, D. Gunupulos, N. Koudas, and S. Berchtold, "An efficient approximation scheme for data mining tasks," in *Data Engineering, 2001. Proceedings. 17th International Conference on*, pp. 453–462, IEEE, 2001.

[126] G. S. Manku, S. Rajagopalan, and B. G. Lindsay, "Approximate medians and other quantiles in one pass and with limited memory," *SIGMOD Rec.*, vol. 27, pp. 426–435, June 1998.

[127] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Comput. Surv.*, vol. 46, pp. 13:1–13:31, July 2013.

[128] T. Zhang, R. Ramakrishnan, and M. Livny, "Fast density estimation using cf-kernel for very large databases," in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, (New York, NY, USA), pp. 312–316, ACM, 1999.

[129] F. Chen, D. Lambert, and J. C. Pinheiro, "Incremental quantile estimation for massive tracking," in *Knowledge Discovery and Data Mining*, pp. 516–522, 2000.

[130] J. Silva and S. S. Narayananl, "Information divergence estimation based on data-dependent partitions," *Journal of Statistical Planning and Inference*, vol. 140, pp. 3180–3198, 2010.

[131] A. Majdara and S. Nooshabadi, "Efficient data structures for density estimation for large high-dimensional data," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, May 2017.

[132] M. Geilke, E. Frank, A. Karwath, and S. Kramer, "Online estimation of discrete densities," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pp. 191–200, IEEE, 2013.

[133] M. Geilke, A. Karwath, and S. Kramer, "A probabilistic condensed representation of data for stream mining," in *Data Science and Advanced Analytics (DSAA), 2014 International Conference on*, pp. 297–303, IEEE, 2014.

[134] S. Davies and A. Moore, "Interpolating conditional density trees," in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 119–127, Morgan Kaufmann Publishers Inc., 2002.

[135] E. Frank and R. R. Bouckaert, "Conditional density estimation with class probability estimators," in *Asian Conference on Machine Learning*, pp. 65–81, Springer, 2009.

[136] J. Kim and C. D. Scott, "Robust kernel density estimation," *Journal of Machine Learning Research*, vol. 13, no. Sep, pp. 2529–2565, 2012.

[137] M. Kristan and A. Leonardis, "Online discriminative kernel density estimation," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, pp. 581–584, IEEE, 2010.

[138] M. Xu, H. Ishibuchi, X. Gu, and S. Wang, "Dm-kde: dynamical kernel density estimation by sequences of kde estimators with fixed number of components over data streams," *Frontiers of Computer Science*, vol. 8, no. 4, pp. 563–580, 2014.

[139] B. Peherstorfer, D. Pflüge, and H.-J. Bungartz, "Density estimation with adaptive sparse grids for large data sets," in *Proceedings of the 2014 SIAM international conference on data mining*, pp. 443–451, SIAM, 2014.

[140] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard, "Density estimation by wavelet thresholding," *The Annals of Statistics*, pp. 508–539, 1996.

[141] K. Wu, K. Zhang, W. Fan, A. Edwards, and S. Y. Philip, "Rs-forest: A rapid density estimator for streaming anomaly detection," in *Data Mining (ICDM), 2014 IEEE International Conference on*, pp. 600–609, IEEE, 2014.